CMP
United Business Media

Ruby • Perl • D • C/C++ • Python • Java • BASIC

#368 JANUARY 2005

Dr. Dobb's JOURNAL

SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

http://www.ddj.com

PROGRAMMING LANGUAGES

Metamodeling with Perl & AMPL

RSS, Ruby, & the Web

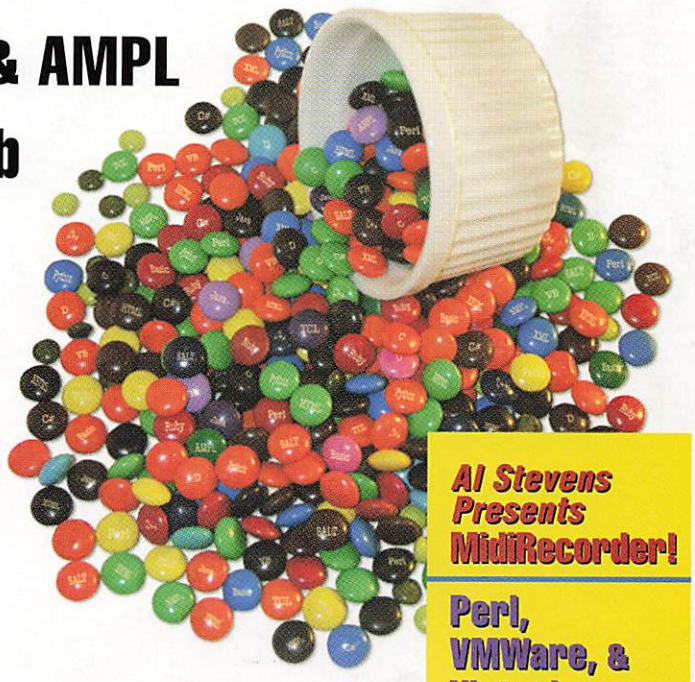
Extending Python

D Revisits *printf*

Voice & XHTML

Cross-Platform Builds

Writing GPS Apps



Al Stevens Presents MidiRecorder!

Perl, VMWare, & Virtual Solutions

Debugging Real-Time Systems

lacl.NET

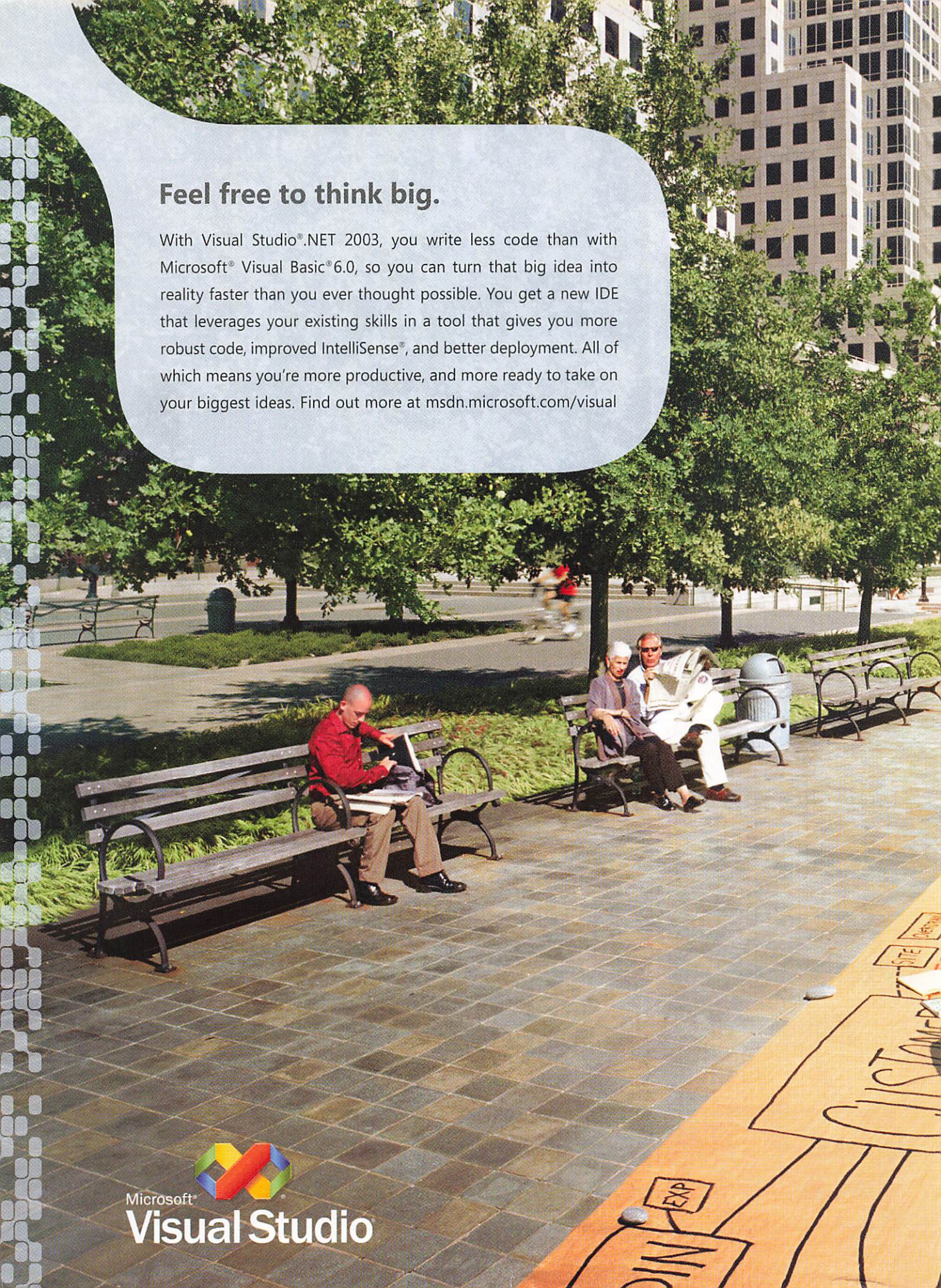
s, Automated Builds, & Visual Studio 2003

More Whidbey C++

$4.95US $6.95CAN

01

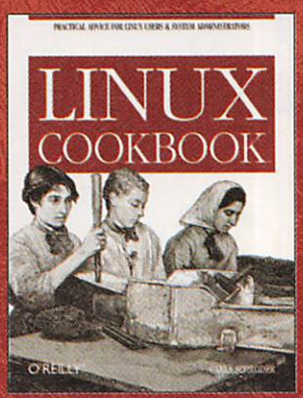
0 74470 01051 7

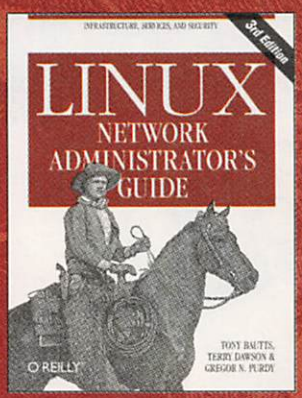

Feel free to think big.
With Visual Studio®.NET 2003, you write less code than with Microsoft® Visual Basic®6.0, so you can turn that big idea into reality faster than you ever thought possible. You get a new IDE that leverages your existing skills in a tool that gives you more robust code, improved IntelliSense®, and better deployment. All of which means you're more productive, and more ready to take on your biggest ideas. Find out more at msdn.microsoft.com/visual
Microsoft
Visual Studio

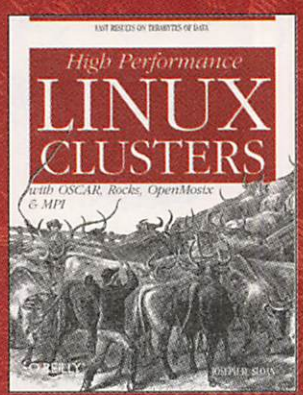
©2005 Microsoft Corporation. All rights reserved. Microsoft, IntelliSense, Visual Basic, Visual Studio and the Visual Studio logo are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

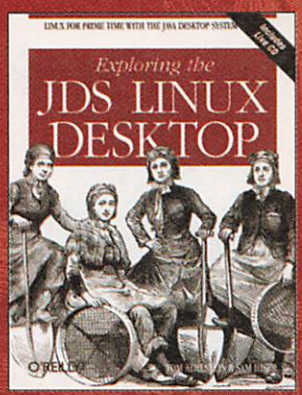
Learn how to give Linux the business.

Your business.









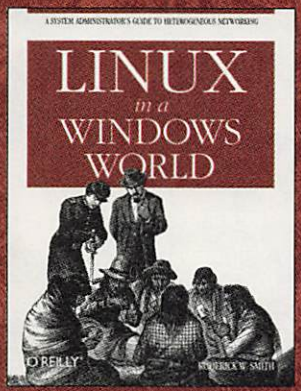
O'REILLY NETWORK

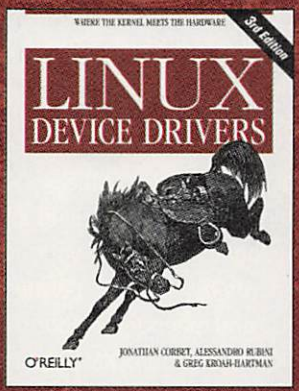
Safari® Bookshelf™

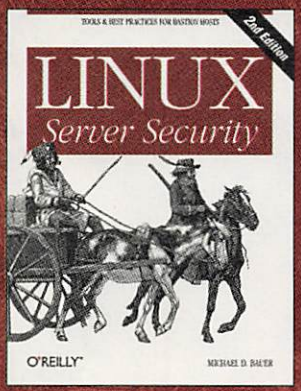
For your complete online Linux library, try the O'Reilly Network Safari Bookshelf. Get free access for 14 days at www.oreilly.com/go/dobbs

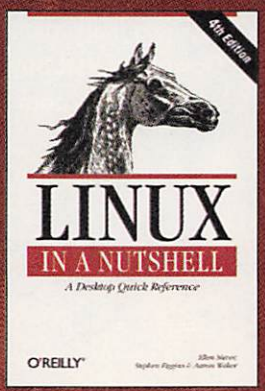
Linux® is among the most stable platforms available for your network servers and business applications, with a track record of security unmatched by others. And, O'Reilly has a decade of experience providing trusted, forward-looking documentation.

O'Reilly offers more tutorials for learning Linux, and hands-on guides for solving specific problems, than any other publisher. Whether you're looking to try Linux on the desktop, or want to move your entire enterprise to the platform, our clear and objective advice will get you there. Linux runs best with O'Reilly books.









Building Embedded Linux Systems

Exploring the JDS Linux Desktop

High Performance Linux Clusters with OSCAR, Rocks, OpenMosix & MPI

Learning Red Hat Enterprise Linux & Fedora, 4th Edition

Linux Cookbook

Linux Device Drivers, 3rd Edition (COMING IN FEBRUARY)

Linux in a Nutshell, 4th Edition

Linux in a Windows World

Linux iptables Pocket Reference

Linux Network Administrator's Guide, 3rd Edition (COMING IN FEBRUARY)

Linux Pocket Guide

Linux Security Cookbook

Linux Server Hacks

Linux Server Security, 2nd Edition

Linux Unwired

Running Linux, 4th Edition

SELinux: NSA's Open Source Security Enhanced Linux

Understanding the Linux Kernel, 2nd Edition

O'REILLY®

www.oreilly.com

© 2004 O'Reilly Media, Inc. O'Reilly logo is a registered trademark of O'Reilly Media, Inc. All other trademarks are property of their respective owners. part#40405

Altova® XMLSpy® 2005 – The industry standard XML development environment.

ALTOVA®
xmlspy®
2005

Bring your development plans to light

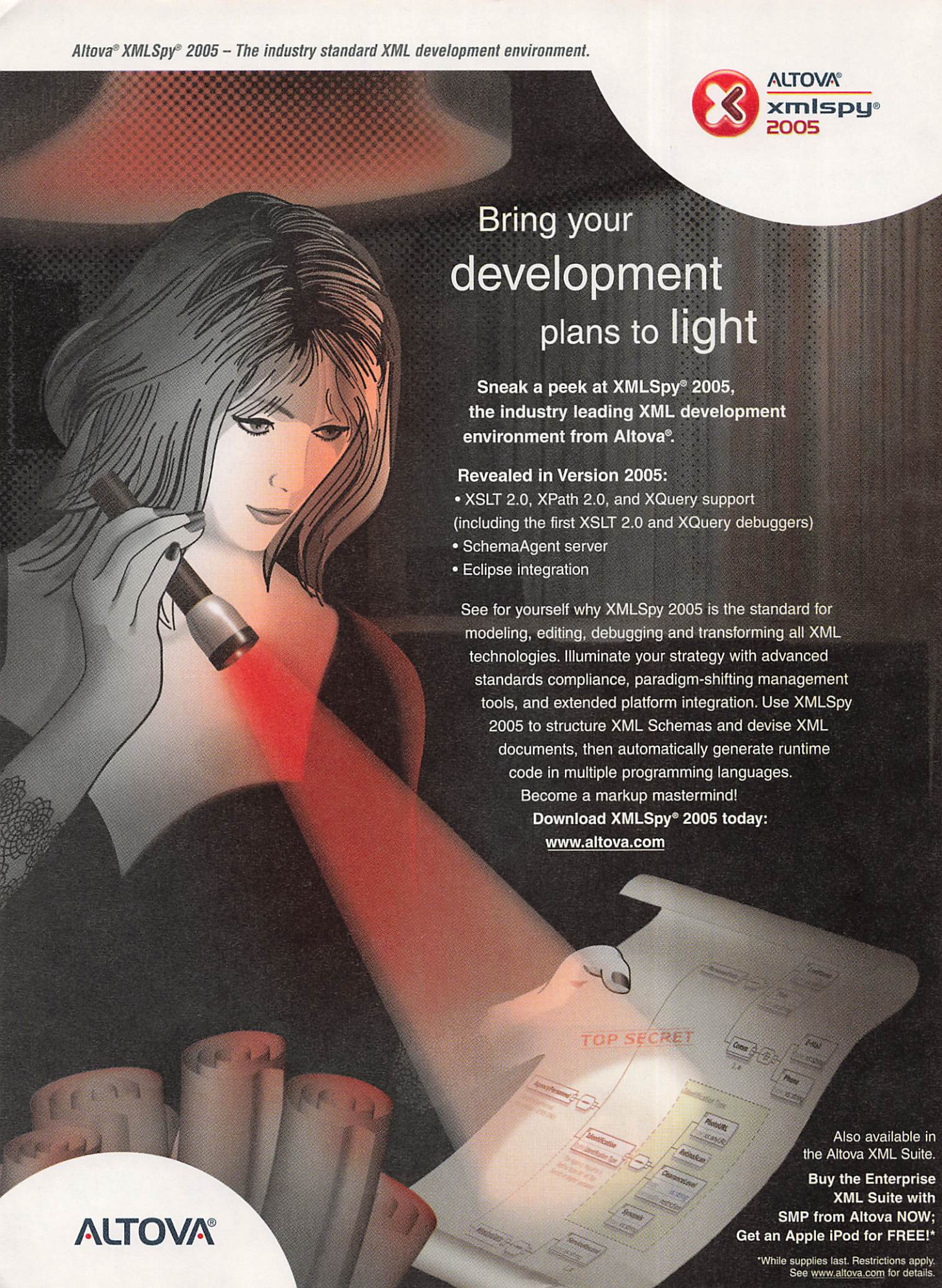
Sneak a peek at XMLSpy® 2005, the industry leading XML development environment from Altova®.

Revealed in Version 2005:

- XSLT 2.0, XPath 2.0, and XQuery support (including the first XSLT 2.0 and XQuery debuggers)
- SchemaAgent server
- Eclipse integration

See for yourself why XMLSpy 2005 is the standard for modeling, editing, debugging and transforming all XML technologies. Illuminate your strategy with advanced standards compliance, paradigm-shifting management tools, and extended platform integration. Use XMLSpy 2005 to structure XML Schemas and devise XML documents, then automatically generate runtime code in multiple programming languages. Become a markup mastermind!

Download XMLSpy® 2005 today:
www.altova.com



ALTOVA®

Also available in the Altova XML Suite.

Buy the Enterprise XML Suite with SMP from Altova NOW; Get an Apple iPod for FREE!*

*While supplies last. Restrictions apply. See www.altova.com for details.

CONTENTS

JANUARY 2005 VOLUME 30, ISSUE 1

FEATURES

Metamodeling with Perl and AMPL 16
by Christian Hicks and Dessislava Pachamanova
To create sophisticated financial models that required simulation, statistical analysis, and multistage optimization, our authors turned to AMPL and Perl.

RSS, Ruby, & the Web 26
by Dave Thomas
Ruby 1.8.2 comes with a full set of libraries that make working with the Web and RSS easy.

Extending Python 32
by Greg Smith
One of Python's greatest strengths is that you can extend it with modules written in C or C++.

Mixins Without Multiple Inheritance 37
by Joe Strout
Mixin classes blend with other classes through multiple inheritance. Joe implements mixins for REALbasic.

***printf* Revisited 40**
by Walter Bright
The D programming language addresses some shortcomings of C and C++'s venerable *printf*.

Adding Voice to XHTML 42
by Gerald McCobb and Jeff Kusnitz
XHTML+Voice is a multimodal markup language that enables voice interaction with web applications.

Cross-Platform Builds 48
by John Graham-Cumming
John presents a cross-platform pattern for Makefiles that works across Windows and common UNIX platforms.

Writing GPS Applications 52
by Jon Person
For commercial-grade GPS apps to work with different devices, you must understand how GPS data is interpreted.

MidiRecorder: An Exercise in Code Reuse 57
by Al Stevens
Code reuse can mean less code in your programs, and more money in your tip jar.

Perl, VMWare, & Virtual Solutions 60
by Sam Tregar
The Krang Farm is an automated build and test system created using VMWare and Perl.

Jacl.NET 81
by Will Ballard
Jacl.NET is a port of the Jacl Java-based TCL interpreter to the .NET Framework.

Whidbey C++ Syntax 86
by Richard Grimes
Richard continues his examination of Whidbey C++, covering new compiler and linker switches, and the build process.

Macros, Automated Builds, & Visual Studio 2003 S12
by Jake Watkins
You can extend the Visual Studio 2003 object model by writing macros in Visual Basic.

EMBEDDED SYSTEMS

Using Trace to Debug Real-Time Systems 66
by Michael Lindahl
Hardware trace data gives you full *printf()* debugging in a completely nonintrusive fashion.

COLUMNS

Programming Paradigms 70
by Michael Swaine

Embedded Space 73
by Ed Nisley

Chaos Manor 76
by Jerry Pournelle

Programmer's Bookshelf 81
by Matthew Wilson

Cover photography by Margaret Anderson.

FORUM

EDITORIAL 8
by Jonathan Erickson

LETTERS 10
by you

Dr. Ecco's Omniheurist Corner 12
by Dennis E. Shasha

NEWS & VIEWS 14
by Shannon Cochran

OF INTEREST 82
by Shannon Cochran

SWAINE'S FLAMES 84
by Michael Swaine

RESOURCE CENTER

As a service to our readers, source code, related files, and author guidelines are available at http://www.ddj.com/. Letters to the editor, article proposals and submissions, and inquiries can be sent to editors@ddj.com, faxed to 650-513-4618, or mailed to *Dr. Dobb's Journal*, 2800 Campus Drive, San Mateo CA 94403.

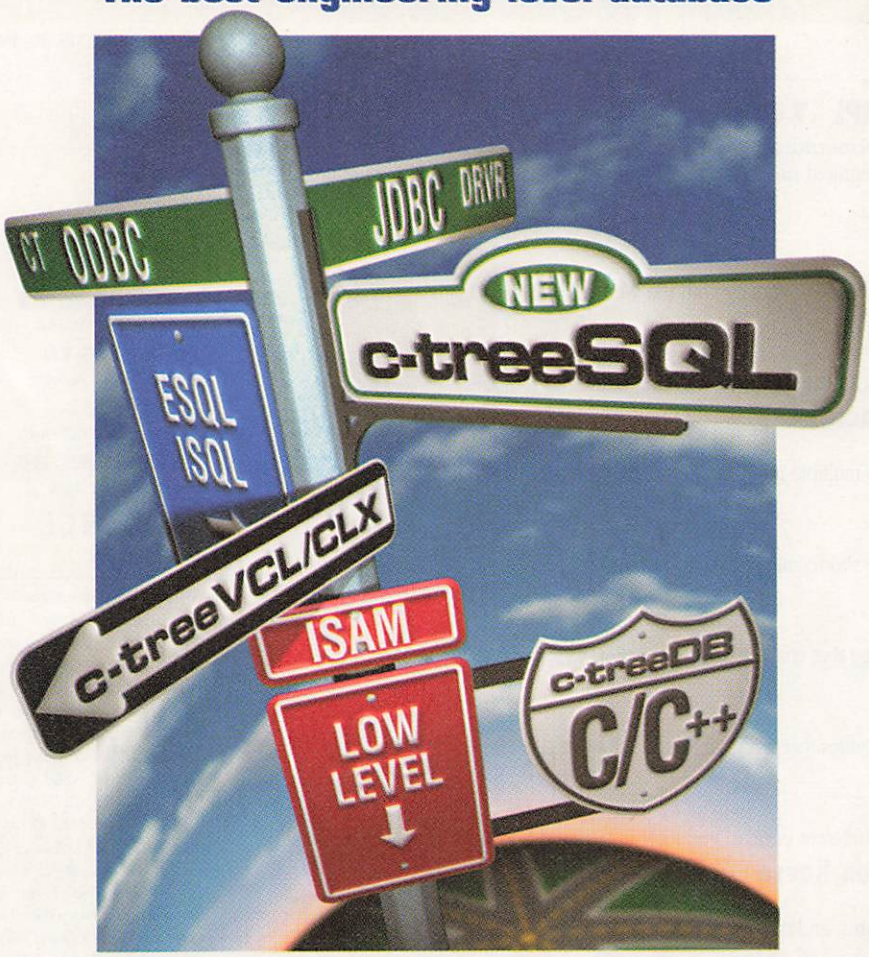
For subscription questions, call 800-456-1215 (U.S. or Canada). For all other countries, call 902-563-4753 or fax 902-563-4807. E-mail subscription questions to ddj@neodata.com or write to *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80322-6188. If you want to change the information you receive from CMP and others about products and services, go to http://www.cmp.com/feedback/permission.html or contact Customer Service at the address/number noted on this page.

Back issues may be purchased for $9.00 per copy (which includes shipping and handling). For issue availability, send e-mail to orders@cmp.com, fax to 785-838-7566, or call 800-444-4881 (U.S. and Canada) or 785-838-7500 (all other countries). Back issue orders must be prepaid. Please send payment to *Dr. Dobb's Journal*, 4601 West 6th Street, Suite B, Lawrence, KS 66049-4189. Individual back articles may be purchased electronically at http://www.ddj.com/.

NEXT MONTH: Web services are our focus in February.

DR. DOBB'S JOURNAL (ISSN 1044-789X) is published monthly by CMP Media LLC., 600 Harrison Street, San Francisco, CA 94017; 415-947-6000. Periodicals Postage Paid at San Francisco and at additional mailing offices. SUBSCRIPTION: $34.95 for 1 year; $69.90 for 2 years. International orders must be prepaid. Payment may be made via Mastercard, Visa, or American Express; or via U.S. funds drawn on a U.S. bank. Canada and Mexico: $45.00 per year. All other foreign: $70.00 per year. U.K. subscribers contact Jill Sutcliffe at Parkway Gordon 01-49-1875-386. POSTMASTER: Send address changes to *Dr. Dobb's Journal*, P.O. Box 56188, Boulder, CO 80328-6188. Registered for GST as CMP Media LLC, GST #13288078, Customer #2116057, Agreement #40011901. INTERNATIONAL NEWSSTAND DISTRIBUTOR: Worldwide Media Service Inc., 30 Montgomery St., Jersey City, NJ 07302; 212-332-7100. Entire contents © 2005 CMP Media LLC.
Dr. Dobb's Journal® is a registered trademark of CMP Media LLC. All rights reserved.

The best engineering-level database



Now reach any development destination

Easily navigate your database development challenges with c-tree's versatile interfaces. These new inroads into our proven core technology give you the flexibility and performance that distinguishes c-tree Plus without incurring significant overhead. Easily mix and match interfaces within your application for maximum control!

c-treeSQL – a powerful new SQL interface that includes embedded SQL and interactive SQL as well as server-side ODBC and Type IV JDBC drivers

c-treeDB – new simplified C and C++ APIs giving convenient access into our proven core

c-treeVCL/CLX – new data access components for Delphi™, C++ Builder™, and Kylix™ visual development tools

ISAM/Low-level – our traditional C APIs that have driven our success for almost 25 years

check out our FREE white paper "Using c-tree & ADO in your Client/Server Application" at www.faircom.com/ep/ddj/ado

FairCom®

USA • Europe • Japan • Brazil

Other company and product names are registered trademarks or trademarks of their respective owners. © 2003 FairCom Corporation

Dr. Dobb's JOURNAL
SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

PUBLISHER *Michael Goodman*
EDITOR-IN-CHIEF *Jonathan Erickson*

EDITORIAL
MANAGING EDITOR
Deirdre Blake
MANAGING EDITOR, DIGITAL MEDIA
Kevin Carlson
SENIOR PRODUCTION EDITOR
Monica E. Berg
NEWS EDITOR
Shannon Cochran
ASSOCIATE EDITOR
Della Wyser
ART DIRECTOR
Margaret A. Anderson
SENIOR CONTRIBUTING EDITOR
Al Stevens
CONTRIBUTING EDITORS
Bruce Schneier, Ray Duncan, Jack Woehr, Jon Bentley, Tim Kientzle, Gregory V. Wilson, Mark Nelson, Ed Nisley, Jerry Pournelle, Dennis E. Shasha
EDITOR-AT-LARGE
Michael Swaine
PRODUCTION MANAGER
Douglas Ausejo

INTERNET OPERATIONS
DIRECTOR
Michael Calderon
SENIOR WEB DEVELOPER
Steve Goyette
WEBMASTERS
Sean Coady, Joe Lucca

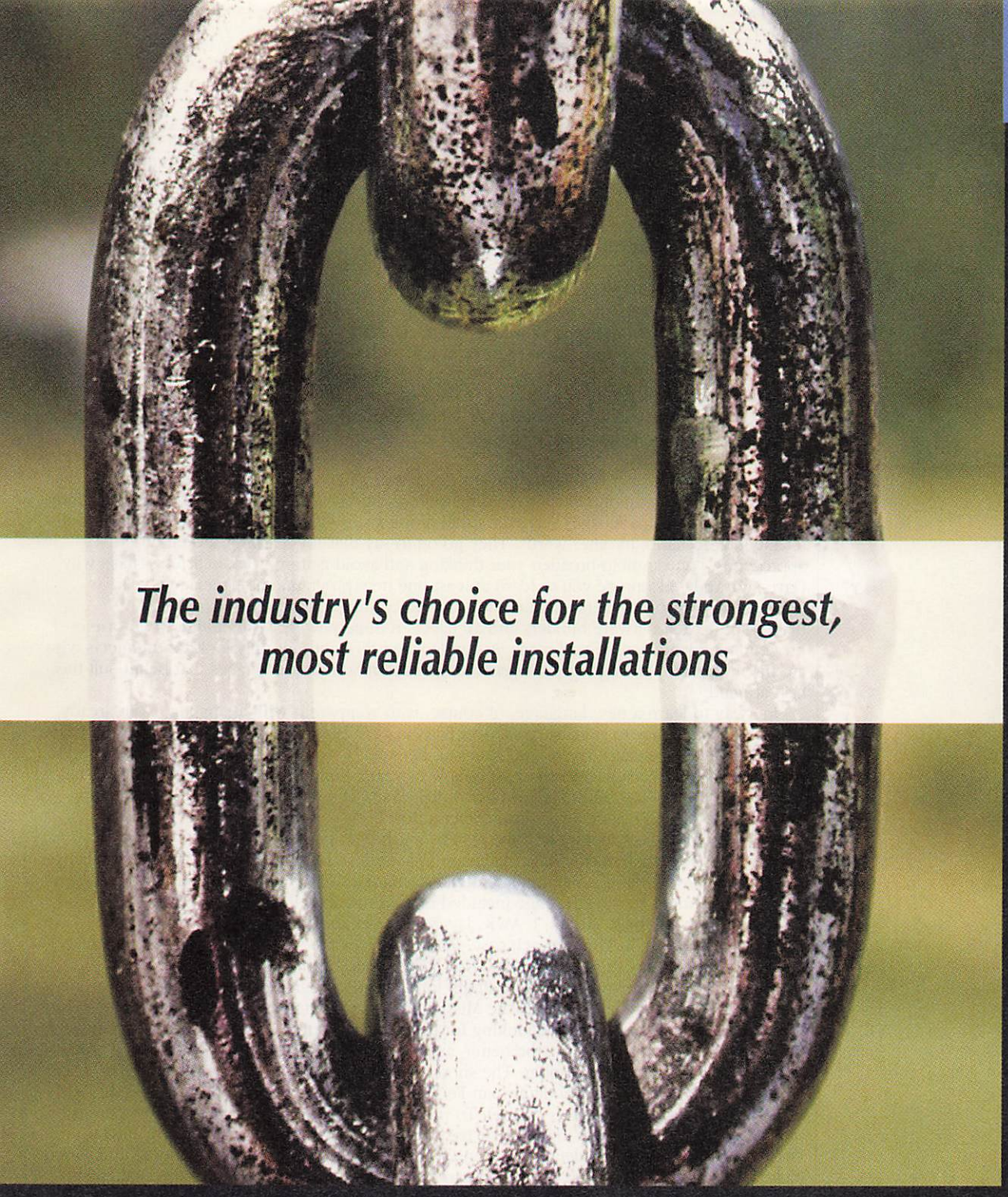
CIRCULATION
SENIOR CIRCULATION MANAGER
Cherilyn Olmsted
ASSISTANT CIRCULATION MANAGER
Shannon Weaver

MARKETING/ADVERTISING
MARKETING DIRECTOR
Jessica Hamilton
AUDIENCE DEVELOPMENT DIRECTOR
Ron Cordek
ACCOUNT MANAGERS see page 83
Michael Beasley, Randy Byers, Andrew Mintz, Erin Rhea, Cassandra Clark, Mike Kelleher
SENIOR ART DIRECTOR OF MARKETING
Carey Perez

DR. DOBB'S JOURNAL
2800 Campus Drive, San Mateo, CA 94403
650-513-4300. http://www.ddj.com/

CMP MEDIA LLC
Gary Marshall President and CEO
John Day Executive Vice President and CFO
Steve Weitzner Executive Vice President and COO
Jeff Patterson Executive Vice President, Corporate Sales & Marketing
Mike Mikos Chief Information Officer
William Amstutz Senior Vice President, Operations
Leah Landro Senior Vice President, Human Resources
Mike Azzara Vice President/Group Director Internet Business
Sandra Grayson Vice President & General Counsel
Alexandra Raine Vice President Communications
Robert Faletra President, Channel Group
Vicki Masseria President CMP Healthcare Media
Philip Chapnick Vice President, Group Publisher Applied Technologies
Michael Friedenberg Vice President, Group Publisher InformationWeek Media Network
Paul Miller Vice President, Group Publisher Electronics
Fritz Nelson Vice President, Group Publisher Network Computing Enterprise Architecture Group
Peter Westerman Vice President, Group Publisher Software Development Media
Joseph Braue Vice President, Director of Custom Integrated Media Solutions
Shannon Aronson Corporate Director, Audience Development
Michael Zane Corporate Director, Audience Development
Marie Myers Corporate Director, Publishing Services

CMP United Business Media
ABP American Business Press
Printed in the USA
BPA



NEW VERSION

InstallShield®

The industry's choice for the strongest, most reliable installations

InstallShield 10.5, now with XML configuration tools, Trialware functionality, and MSI 3.0 support to make your setups the strongest and most reliable link between your product and your customers.

Microsoft CERTIFIED Partner

Download Now!

www.installshield.com/10.5_strong

Special Offer!

Receive expert MSI or InstallScript™ training with purchase. See Web site for details.

macrovision

©1990-2004 Macrovision Europe Ltd. and/or Macrovision Corporation. All Rights Reserved.

InstallShield, a Macrovision Company

The Lure of Languages

Programming languages are the biscuits-and-gravy of software development. Of course, if all you do is follow the tabloids, you'd think that when it comes to programming, there's only something called "C++," another something called "Java," and something that used to be called "COBOL." In truth, there aren't just three or four languages, but hundreds of them. Why, in this issue alone, we cover 15 or so languages, ranging from the familiar (Java, C++, Perl) to the relatively arcane (D, AMPL, and Jcl), with a whole lot in between (Ruby, Python, C#).

Of course, many languages are closely related to each other, sharing similar vocabularies and grammatical rules, not to mention monikers. In the C realm, for instance, there's C, C++, Ch, SystemC, Cg, Cyclone, C99, Objective-C, MeldC, DSP-C, and C++/CLI—and that's just for starters. There's also Coffee, Cocoa, and Curl, although the most they share with other C-variant languages is the first letter of their names. And if that's not enough, new languages pop up every day. Recent entries include Fabl, Comega (also called "Cω"), Xen, Groovy, Pluk, Euler, Heron, Scala, MPD, Mila, and more. Whew! Shoot, former *DDJ* technical editor David Betz ("the king of little languages") used to crank out languages like XLisp, XScheme, Bob, and Drool just for fun.

There are any number of reasons for this neverending interest in new programming languages, foremost among them, according to Pragmatic Programmers Dave Thomas and Andy Hunt, that "different languages solve the same problems in different ways." The old "right tool for the job" scenario, in other words. They go on to say that "by learning several different approaches, you can help broaden your thinking and avoid getting stuck in a rut." That's why Dave and Andy encourage you to learn at least one new programming language a year.

Experienced programmers already know this. According to a study we conducted, 97 percent of master software builders and architects routinely work with multiple (two or more) programming languages. Good managers understand this, and not only look for architects who are fluent in multiple languages, but encourage exploration into languages that better suit the task at hand.

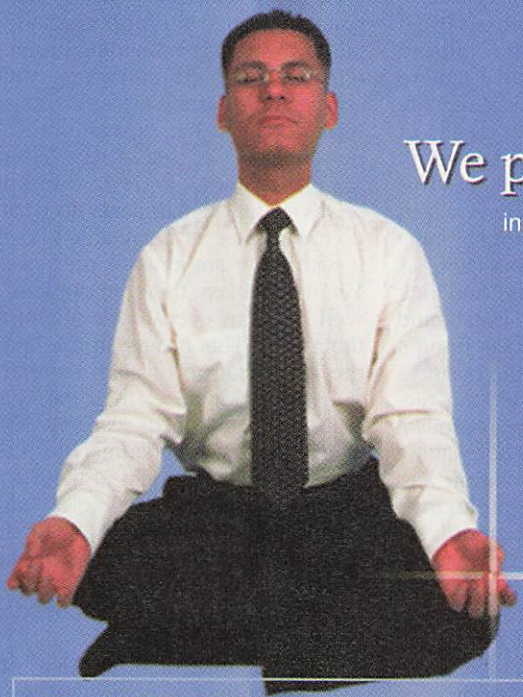
One way to learn a new language, of course, is to compare it with a language you already know. And one place to start with is Oliver Schade's "99 Bottles of Beer: One Program in 621 Variations" web site (http://www.99-bottles-of-beer.net/). Arranged alphabetically from A+ to ZZT, Oliver has collected, over the past couple of years, hundreds of implementations of that awful school bus song "99 Bottles of Beer on the Wall." If you'd like to make a contribution for, say, the W or EPL languages, drop Oliver a note.

Likewise, Frank Buß reported on the results of a programming challenge he recently devised (http://www.frank-buss.de/challenge/index.html). In a nutshell, his Triangle Challenge required that you write a program that counts all triangles in a figure. Interestingly, the submitted solutions to the challenge included implementations in Lisp, Ruby, Java, C, C++, Haskell, Scheme, Python, Kogut, J, AWK, JavaScript, Matlab, SQL, SETL2, D, Prolog, and OCaml—but mostly in Lisp. In each case, Frank provides the source code (with commentary) so that you can compare not only, say, Lisp to Ruby, but Lisp to Lisp.

In this spirit, we're teaming up with Frank for a second programming challenge—this one focusing on a hypothetical Mars Rescue Mission (http://www.frank-buss.de/marsrescue/index.html). In this case, we'll be looking for the shortest solution. The programming language is up to you. The more esoteric, the better, as far as I'm concerned, although that may only get you brownie points. Remember: The shortest path from A to Z is what counts. All solutions must be submitted no later than February 1, 2005, and we'll publish the results in our Algorithms issue (as well as online). To make it interesting, the top five submissions will receive copies of *Dr. Dobb's* CD/Release 16, which includes archives from January 1988 to December 2004. Again, for details see http://www.frank-buss.de/marsrescue/index.html. Good luck. Who knows, maybe you'll even have to create a new language to solve the challenge.

Jonathan Erickson
editor-in-chief
jerickson@ddj.com

AMYUNI Technologies



We personalize your needs...

in document creation and conversion for
PDF, RTF, DHTML, EXCEL®, JPEG and TIFF™ formats!

Featured Product
PDF Converter
Version 2.5

New Features In Version 2.5

- Applying digital signatures to PDF documents
- Embedding of Type1(Postscript) fonts in the PDF file
- Generation of CMYK colored PDF files
- Automatic generation of bookmarks and hyperlinks
- Direct link to document management systems such as iManage
- Improved support for Office XP/2003
- Improved support for third party PDF files for printing, merging or appending

AMYUNI PDF CONVERTER
The PDF Converter enables you to create secure PDF documents from any Windows® application. The full featured DLL, ActiveX or .NET interface gives you complete control over the PDF creation process. You can also print, optimize, append, merge, email and do a lot more with your PDF files.

AMYUNI PDF CREATOR
The PDF Creator ActiveX control enables you to create secure PDF documents on the fly and to view, print and process existing PDF documents. Additional plugins also enable you to save your documents in RTF, HTML, EXCEL®, JPEG or TIFF™ formats.

AMYUNI DOCUMENT CONVERTER SUITE
PDF-RTF-HTML-Excel® -JPEG-TIFF™
Amyuni's new Document Converter gives you more document conversion power in a more streamlined product. Our flagship PDF Converter technology has been applied to RTF, HTML, Excel, JPEG and TIFF™ formats and integrated them in one global Amyuni printer driver.

USA & CANADA
Toll Free: 1-866-926-9864
Support: (514) 868-9227

Europe
Sales: (+33) 1 30 61 07 97
Support: (+33) 1 30 61 07 98
info: sales@amyuni.com
Customizations: management@amyuni.com

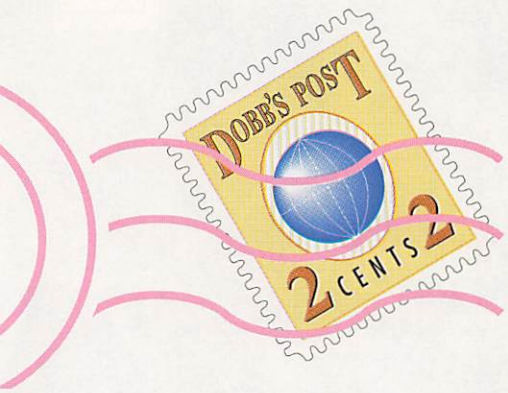
Available licenses:
Single-User
Server
Royalty-free Developer
Unlimited Corporate
OEM

For complete evaluations, technical notes and online purchasing visit: **www.amyuni.com**

Our priority is customer satisfaction!

www.amyuni.com

All trademarks are property of their respective owners.© 1999-2005 AMYUNI Technologies All rights reserved.



Licensing Requirements

Dear *DDJ*,

Jonathan Erickson's June 2003 "Editorial" was pretty good, but I thought no more of the matter until I read Brent Fulgham's formidable counter in the August *DDJ*.

If a licensing requirement is imposed upon software practitioners in the manner described, we might as well amend Brent's Fulgham's statement in his second paragraph to:

> Most of us take for granted that the bridges we drive on, the airplanes we fly in, the medicines we use in our lives, and the Space Shuttles we as taxpayers pay for are safe, reliable, and cost effective, the products of careful design and peer review.

Some may think I am being unfair, but I'm not. First, most people in information technology today do not design or write code from the ground up, as Brent seems to think. We write code to work with other software and software systems, adapt them, or enhance them. It is a mark of early 21st century software practice that we have no specifications for these systems. Those we do have, even if informal, are often inconsistent with how the software actually behaves. That behavior is discovered in debugging our own. Since any practical licensing requirement will necessarily grandfather existing software, holding new code to the more stringent requirement is ridiculous.

Second, I do not know where Brent has been for 10 years, but the software industry has transformed in that time to one where cost and speed to delivery are paramount, and the primary means of achieving that has been to adopt shrinkwrap packages everywhere. This means businesses and institutions, which once employed their own programming staff, no longer have much of a DP or IT staff—they're called "analysts," and their jobs are to specify, procure, install, and superficially maintain products bought off the shelf. This has been the business response and solution to the long recognized problem of software reliability. Business people got impatient awaiting the fruits of software innovation and found their preconditions intolerable. They fixed the problem by amortizing the cost of development over many customers and adapting their practices and business policies to the limited capabilities of existing packages. Software is unpredictable and it is simply not something most of them consider near or pertinent to their core business. Where custom software is needed, outsourcing is the new way, including outsourcing using non-U.S. teams. Requiring licenses of software professionals in the U.S. will simply accelerate this trend. Brent and the Texas Society of Professional Engineers don't seriously expect they can impose this requirement on *all* software used in the U.S., irrespective of origin, do they?

Third, as attractive as many of the high-precision techniques for designing and developing software are, such as Design By Contract and similar methods, they are inadequate because they expect detailed and accurate specifications to be available for software to be developed. While there are methods available in approaches like Extreme Programming for mitigating some of these demands, business people, end users, and functional users are simply not used to working this way. When building business intensive, detailed software, such as enterprise reporting systems or systems which demand the explication of "business rules," it is clear, these people do not understand or know how they themselves or their departments do what they do in the sense of being able to describe it with large amounts of precise detail. Yet, nonetheless, they can do these tasks well and can even teach others to do them well. This means, in the end, that the specification of such software requires developers or their surrogate to adjust its function in response to end-user critiques, complaints, and bug reports, even if the origin of their dissatisfaction has nothing at all to do with the software's implementation, being, for instance, a result of incorrect expectations or documentation regarding the nature of, say, data being fed it from some remote database.

Under these conditions, ideas of software blueprints, precise legal accountability, or even of standards of quality are meaningless. The edifices are being built on shifting sands. This is not a desirable set of circumstances and I am not trying to argue that they are. But it is what business and users feel they need to be successful and, if we want to be successful practitioners of the software art, we need to understand its limitations and serve them using what's there and what we know. Demanding some hypothetical and unrealistic set of terms and conditions will not do it.

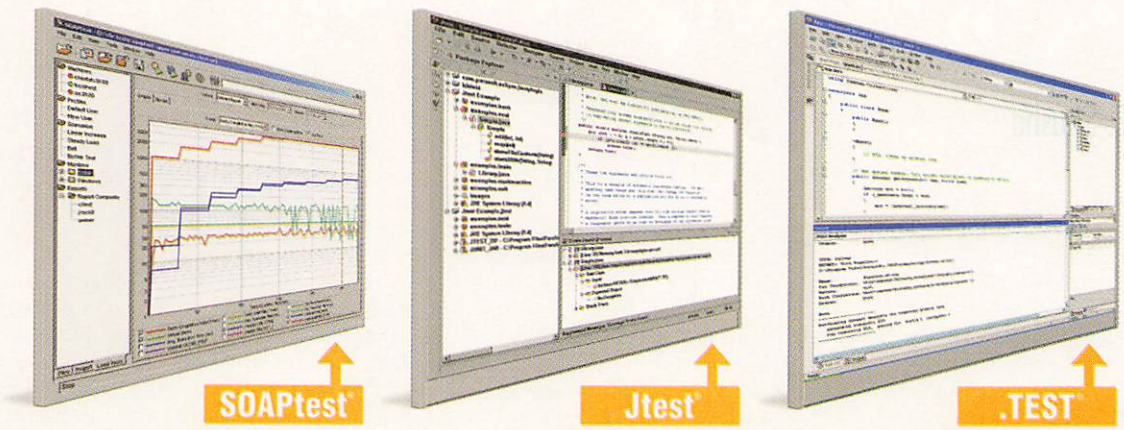
Jan Theodore Galkowski
disneylogic@fastmail.fm

DDJ

Come to the Rescue!
(And Win a Prize)
Pit your programming skills against your peers in *Dr. Dobb's Journal* Mars Rescue Mission Challenge!
ANY PROGRAMMING LANGUAGE—IT'S THE ALGORITHM THAT COUNTS!
http://www.frank-buss.de/marsrescue/index.html

Achieve higher quality

in less time, with fewer resources.



Parasoft® Automated Error Prevention (AEP) Products:

Specifically designed for under-staffed, over-committed development organizations like yours.

Parasoft AEP Products ensure that comprehensive error prevention practices are applied consistently and uniformly across your entire team.

By automating compliance to a unified set of testing and coding standards, Parasoft AEP Products enable every team member to follow the same best practices and the same error prevention techniques – including coding standards analysis, unit testing, code review and regression testing.

Automated testing and standards compliance for any team configuration.

Parasoft AEP Products automate error prevention for Java, C/C++, Web Services, the Microsoft® .NET Framework, Embedded Development, Web Development, Database Development and more.

For details go to: www.Parasoft.com/AchieveQuality

Call: 888-305-0041 x3307 Email: AchieveQuality@Parasoft.com

PARASOFT®
We make software work.

Founded in 1987, Parasoft's clients include IBM, HP, DaimlerChrysler and over 10,000 companies worldwide.

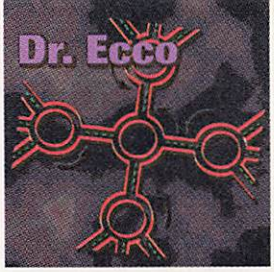
Availability

Parasoft AEP Products are available on Linux, Solaris, and Windows NT/2000/XP.

Contact Parasoft for details and pricing information.

Parasoft Corporation
101 E. Huntington Dr., 2nd Flr.,
Monrovia, CA 91016

Copyright©2004 Parasoft Corporation. All rights reserved. All Parasoft product names are trademarks or registered trademarks of Parasoft Corporation in the United States and other countries. All others marks are the property of their respective owners.



The Luck of the 4

Dennis E. Shasha

Ecco encouraged his niece and nephew to bring over their friends whenever they wanted. Most people would have considered Camille and Maia to be very unusual teenagers. Naturally, they fit right into the Ecco household. Good-looking girls with a fine sense of music, they accompanied Liane's singing and guitar strumming using Tyler's drums.

"Our England-born Dad heard this game from a school teacher back in the day," Camille told us after the singing had stopped.

"Now it's going to give us a hint about a secret phone number. The game is called 'four 4s.' The idea is that you want to set up an arithmetic expression to give you as many consecutive integers as possible using the digit 4 alone, at most four times. For example, you can get 1 by dividing 4 by itself and 2 by taking the square root of 4.

"The expression may include the following operators: addition, subtraction, multiplication, division, square root, exponentiation (for example, 4^4), concatenation (44, which uses 4 twice), decimal 0.4, decimal repeat 0.4R=0.4444444444...= 4/9 (this uses 4 once), factorial (4!=24, for example), and parentheses (to distinguish (4+4)/4 from 4+(4/4))."

Dennis is a professor of computer science at New York University. His latest books include Dr. Ecco's Cyberpuzzles: 36 Puzzles for Hackers and Other Mathematical Detectives *(W.W. Norton, 2002) and* Database Tuning: Principles, Experiments, and Troubleshooting Techniques *(Morgan Kaufman, 2002). He can be contacted at DrEcco@ddj.com.*

Warm-Up: It is possible to find expressions for the numbers 1 through 10 under these constraints. If you try really hard, you can use three 4s or fewer for every number. Try it before you read on.

Solution to Warm-Up:

1=4/4
2=sqrt(4)
3 =sqrt(4/.4R) = sqrt(4/(4/9)) = sqrt(9)
4=4
5=sqrt(4)/0.4
6=4 + sqrt(4)
7=4/.4R – sqrt(4)
8=4*sqrt(4)
9=4/.4R
10=4/.4

"We have had trouble going much beyond this. We believe that the first numbers that can't be expressed in terms of four or fewer 4s (we call those "gaps") give us a good hint as to the 10-digit U.S. secret phone number. Are there any gaps below 40?"

"I don't think so," said Tyler proceeding to show encodings for those numbers.

"Nice use of factorial," Maia said smiling.

Tyler found expressions for 11 to 40 using at most four 4s per integer. All but one of the numbers below 30 could be constructed using three 4s or fewer. Can you do as well or better?

The girls left, quite satisfied with their meeting, but Liane was quite annoyed.

"Tyler, we should have been able to help them more. Here are some open problems:

"Open 1. How far can we go under these rules?

"Open 2. How many of the integers between 1 to 1000 can be handled with just four 4s? What are the first gap numbers? Do any of those gaps make sense as phone numbers?

> "The idea is that you want to set up an arithmetic expression to give you as many consecutive integers as possible using the digit 4 alone at most four times"

"Open 3. If we were allowed five 4s, what is the longest sequence of consecutive integers we could produce? (It doesn't have to start at 1.)"

Tyler smiled and said, "I've worked on the four 4s question up to 100. Beyond that I don't know."

I had to leave before seeing his answer. It's now open season on the open questions.

For the solution to last month's puzzle, see page 72.

DDJ

Introducing tools you'll actually use.

We've all bought our share of shelfware. Seduced by visions of breakthrough performance and eye-popping results, we gleefully loaded the latest software, only to find it cumbersome and overly complicated. And so it was successfully installed ... on the shelf.

Enerjy software tools for Java: Simple. Unobtrusive. And refreshingly useful.

At Enerjy, we believe that the purpose of software tools is to help developers create better applications faster, without getting in the way. That's why we designed our tools to integrate seamlessly within your IDE. They let you analyze your code, monitor performance, and identify thread problems and memory leaks—all without breaking your stride.

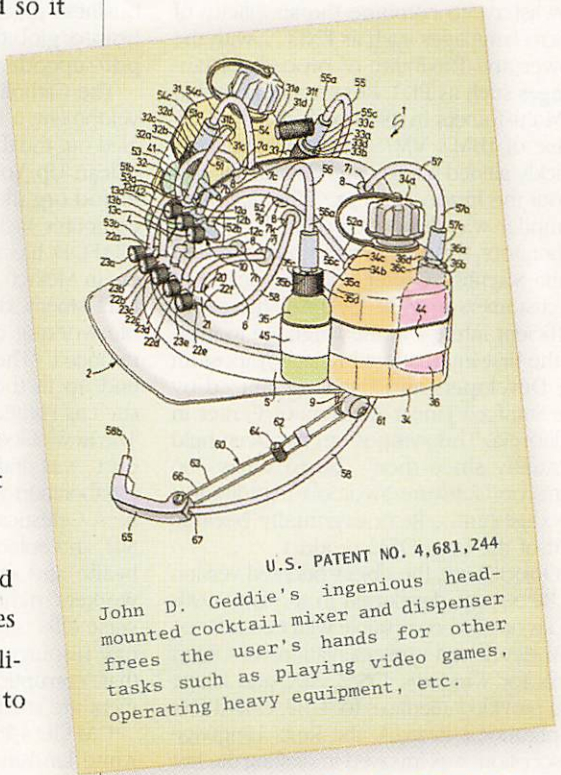
Sure, we can trot out ROI studies, user stories, and application briefs that prove our software increases productivity, reduces development cycles, and reliably produces eye-popping results. But we'd prefer to let you draw your own conclusions.

So go to **www.enerjy.com** and add Enerjy to your Java development today. If our tools don't become an indispensable part of your development process, return them for a full refund under our Anti-Shelfware Guarantee. Hey, your shelves are crowded enough as it is.

Visit our Web site to learn more about our Java tools:

ENERJY | CODE ANALYZER	Best practices audit tool
ENERJY | PERFORMANCE PROFILER	Improve application speed
ENERJY | MEMORY PROFILER	Reduce your memory footprint
ENERJY | THREAD PROFILER	Find, fix, and avoid thread problems

© 2004 Enerjy Software. All rights reserved. Enerjy and Enerjy Software are trademarks of Teamstudio, Inc. Java is a trademark of Sun Microsystems, Inc.



U.S. PATENT NO. 4,681,244

John D. Geddie's ingenious head-mounted cocktail mixer and dispenser frees the user's hands for other tasks such as playing video games, operating heavy equipment, etc.

From the "Tools You're Not Likely Ever To Use" archive.

ENERJY™

Java software tools

www.enerjy.com

IBM Open Sources Object Rexx

IBM has released Object Rexx to the nonprofit group RexxLA (http://www.rexxla.org/), which will manage Object Rexx as an open-source project under the Common Public License.

Rexx was designed by IBM's Mike Cowlishaw to combine the simplicity of macro languages such as EXEC 2 with the power and flexibility of procedural languages such as PL/I. Rexx was shipped to IBM customers in 1983 with the third release of IBM's VM/System Product and quickly gained a following. Cowlishaw says about the history of Rexx, "The first Rexx compiler was developed at IBM's Vienna Laboratory, following research by the IBM Haifa Scientific Center, and was delivered to customers in 1989. By 1990, there was sufficient interest in the language to justify the first international Rexx Symposium for Developers and Users, organized by the Stanford Linear Accelerator Center in California. This symposium has been held annually since then" (http://www-306.ibm.com/software/awdtools/rexx/library/rexxhist.html). Rexx eventually became part of the base OS/2 product.

Object Rexx, the object-oriented version of Rexx, was developed in the late 1990s and is upward-compatible with classic Rexx. IBM developed implementations of Object Rexx for Windows, OS/2, AIX, and Linux, and provided interfaces for DB2, C, and C++ applications. RexxLA, the Rexx Language Association, was involved in drafting the first ANSI standard for Rexx in 1996. The group is now working with ANSI to define a Standard for Object Rexx and NetRexx (a version of Rexx for the Java platform).

Ethical "Code"

In the wake of a report charging that their overseas workers "suffer harassment, discrimination, and intolerable working conditions," Hewlett-Packard, IBM, and Dell have pledged to abide by a code of conduct respecting workers' rights, environmental standards, and fair business practices (http://www.hp.com/hpinfo/globalcitizenship/environment/pdf/supcode.pdf).

The Catholic Agency for Overseas Development, a British nonprofit group, singled out the three companies in its report, "Clean Up Your Computer" (http://www.cafod.org.uk/news_and_events/news/computer_factory_sweatshops_20040126). "CAFOD has proof that electronic workers in Mexico, Thailand, and China suffer harassment, discrimination, and intolerable working conditions," the agency announced. "The workers produce parts that end up in the computers of companies such as Hewlett-Packard, Dell, and IBM." The new "Electronic Industry Code of Conduct" was drafted by HP, IBM, and Dell in collaboration with their manufacturing partners: Celestica, Flextronics, Jabil, Sanmina SCI, and Solectron. Along with setting out health and safety standards and a list of workers' rights, the code states that "adverse effects on the environment and natural resources are to be minimized" and that "corruption, extortion, and embezzlement are strictly prohibited."

CAFOD's Private Sector Policy Analyst Anne Lindsay responded in a prepared statement: "What matters is improving the situation on the factory floor. There's not much detail in the code on how change will actually be achieved. We'll be looking to the participating companies for evidence that their suppliers meet the standards set out here."

First Anita Borg Awards

Dr. Fran Allen of IBM and Karen Banks, coordinator of the APC Women's Networking Support Program in London, are the recipients of the first Anita Borg Awards for women in technology (http://www.anitaborg.org/). The newly founded awards are designed to be bestowed in pairs; one honors technical leadership, and the other social impact. Each is accompanied by a $10,000 prize.

The first Anita Borg Award for Technical Leadership, underwritten by Greg Papadopoulos, was given to Allen for her work on the theory of program optimization, for leading the PTRAN (Parallel Translations) project, and for mentoring other researchers. Allen was the first woman to be named an IBM Fellow; she was also elected president of the IBM Academy of Technology and has been a member of the National Academy of Engineering. She has been inducted into the Women in Technology International Hall of Fame, and she has also received the ACM SIGPLAN's Programming Languages Achievement Award. The Anita Borg Award for Social Impact, underwritten by Microsoft, went to Karen Banks, the global Coordinator for APC Women's Networking Support Program for working "successfully around the globe to bring the use of Information Communication Technologies to underserved women and communities as a tool for women's empowerment, gender equality, social action and positive social change."

Come to the Rescue!
(And Win a Prize)
Pit your programming skills against your peers in *Dr. Dobb's Journal* Mars Rescue Mission Challenge!
ANY PROGRAMMING LANGUAGE — IT'S THE ALGORITHM THAT COUNTS!
http://www.frank-buss.de/marsrescue/index.html

Stay Up to Date With *Dr. Dobb's* E-mail Newsletters

Delivered FREE to your mailbox!

SIGN UP NOW AT: www.ddj.com/maillists/

NEW ⇨ *Dr. Dobb's* Programmer's Bookshelf Newsletter
A monthly newsletter that brings you reviews of the most important computer books, updates on recent releases, recommendations from our resident experts, and excerpts from new and important books that belong on every programmer's bookshelf.

NEW ⇨ *Dr. Dobb's* Windows/.NET Update
Weekly updates on Windows/.NET development. Online columns featuring practical, hands-on advice from industry experts including Dino Esposito, Richard Grimes, Mark M. Baker, and Jason Coombs; plus highlights every month from *Dr. Dobb's Journal* Windows/.NET Special Supplement.

Java Newsletter Edited by Java expert Al Williams, *Dr. Dobb's* Java Newsletter covers all the Java news that's fit to print, including API revisions and releases, commercial and open-source development tools, Java tech tips, and more.

***Dr. Dobb's* Update** is delivered twice a month. The mid-month mailing will highlight recent articles in *Dr. Dobb's Journal* and www.ddj.com. The end-of-month mailing includes source code published in the following month's issue. For example, January's first mailing will include highlights from March issue and the second mailing will include source code published in the March issue.

Active Scripting Programming tips for Active Scripters, plus the latest info from Mark Baker's Active Scripting Q&A.

Linux Digest A monthly compendium that zeros in on the most important Linux discussion. Edited by Steven Gibson.

***Dr. Dobb's* Software Tools**
Keep up to date with new developer products, from SDKs and libraries, to components and compilers.

...Plus More!

Sign up today
for FREE subscriptions
http://www.ddj.com/maillists/

CMP
United Business Media

Dr. Dobb's JOURNAL
SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

Metamodeling with Perl and AMPL

Using little languages to solve big problems

CHRISTIAN HICKS AND DESSISLAVA PACHAMANOVA

In the 1980s, a significant conceptual breakthrough was accomplished in computational optimization with the introduction of universal optimization languages, such as the Algebraic, or Applied, Mathematical Programming Language (AMPL) and the General Algebraic Mathematical System (GAMS). AMPL and GAMS do not solve optimization problems. Instead, they provide the environment for users to describe their models in a standardized way, and call on separate solvers to solve the problem. The results are returned to human modelers via the same software.

Originally developed at AT&T Bell Labs, AMPL is a modeling language for optimization problems. With AMPL, users describe an optimization problem in the AMPL language, after which the AMPL software processes the problem and passes it to a separate optimization solver. AMPL supports numerous commercial and free solvers. For a full listing of available solvers, see http://www.ampl.com/. The Student Edition of AMPL (which limits the number of variables and constraints in the model) is included with *AMPL: A Modeling Language for Mathematical Programming*, Second Edition, by Robert Fourer, David M. Gay, and Brian W. Kernighan (Duxbury Press/Brooks/Cole Publishing, 2002; ISBN 0534388094). Windows, UNIX (including Linux), and Mac OS X versions of the Student Edition and compatible solvers are also available for free download from http://www.ampl.com/. The full version of AMPL can be purchased from ILOG at http://www.ilog.com/. ILOG also sells an AMPL-compatible version of CPLEX, which is one of the most powerful solvers for linear, mixed-integer, and quadratic optimization. Windows and UNIX (including Linux) binaries of AMPL and CPLEX are available. AMPL is also one of the modeling languages supported by the NEOS Server for Optimization (http://www-neos.mcs.anl.gov/neos/). The NEOS server lets users submit optimization problems via the Internet, solves them for free on a NEOS computer, and returns the results to the submitter.

A simple optimization example from the book *AMPL: A Modeling Language for Mathematical Programming* is as follows: A steel mill processes unfinished steel slabs into steel bands and steel coils. It can produce 200 tons of bands per hour and 140 tons of coils per hour, but cannot sell more than 6000 tons of bands per week or 4000 tons of coils per week. When sold, bands profit $25.00 per ton, and coils profit $30.00 per ton. How many tons of bands and how many tons of coils should be produced to maximize profit?

Christian is president and cofounder of Elysium Digital LLC. Dessislava is an assistant professor of operations research at Babson College, MA. They can be contacted at cbhicks@elys.com and dpachamanova@babson.edu, respectively.

"The PerlAmpl module was designed as an object-oriented Perl class"

To solve this problem, you create the two input files for AMPL, like Examples 1(a) and 1(b). Of course, this is a simple problem, and AMPL can handle problems far more complex. But real-world optimization models can have infinite variety. Frequently, they require multiple refinements of the model and data, and therefore, multiple calls to optimization and data-processing software. Some optimization formulations involve solving sequences of subproblems. Other models require, in addition to optimization, the ability to analyze data through statistical techniques (to run regressions, for example), the ability to simulate scenarios, the ability to pass results from one stage of the computation as inputs for the next stages of the problem, and so on. We define such complex models as "optimization metamodels."

We ran into the problem of solving optimization metamodels while creating sophisticated financial models that required simulation, statistical analysis, and multistage optimization. As we sought to devise a good system for handling optimization metamodels, we focused on leveraging AMPL because of its modern design and its support of a wide range of solvers.

Little Languages and Optimization Systems

In a 1995 talk at Princeton University, Brian Kernighan, one of AMPL's creators, described AMPL as a prime example of a "little language." The term "little languages" was introduced for the first time by Jon Bentley (1986), who was at Bell Labs at the same time AMPL was designed. Bentley considered three different approaches to solving the problem of generating certain images:

- Interactive systems (such as PhotoShop) made drawing simple images easy, but did not allow for programmatic generation of complex pictures.
- Subroutine libraries worked well for generating images from big programs, but were awkward for small tasks.
- Little languages provided an attractive option, since they could be powerful enough to generate complex images in an automated way, but limited enough that users could master the syntax for the language in a reasonable amount of time.

Bentley's reasoning applied perfectly to AMPL, which was designed to allow for the easy handling of both simple and complex optimization problems. We used Bentley's reasoning (as well as our own) in creating a system for optimization metamodeling.

Interactive systems were unsuitable because of the complexity of optimization metamodels. The goal of an integrated system was not to help users create simple models—this problem had already been solved by other systems. We placed a premium on supporting complexity and automation.

The little languages approach adopted by AMPL had certainly produced an effective optimization tool, and could be extended to support optimization metamodeling. This would require adding new functionality to the little language for each task that the metamodeler might need; for example, loading data, running simulations, storing and processing results in fully flexible data structures, launching external programs to process intermediate results. In fact, some such capabilities have been added recently: Today, AMPL provides not only an environment for optimization formulations but also tools for data manipulation and database access. For example, AMPL's table declaration lets you define explicit connections between sets, parameters, variables, and expressions in AMPL, and relational database tables maintained by other software, such as Microsoft Access and Excel. In addition, recently introduced command script features let you specify the number of times a model should be rerun, change values of parameters, and specify output format.

The problem with this continuous expansion of AMPL's capabilities is that a metamodel might need a great variety of functionality, to the point that the little language might not be so little anymore. If AMPL were to encompass the entire functionality of C/C++, Perl, or Java, it would be difficult to learn, to say nothing of how difficult it would be to create.

Alternatively, you could allow a little language access to outside resources by providing a mechanism for the invocation of outside code. You can imagine an AMPL program that made calls to other code written in C/C++, Perl, or Java. This option provoked the question: If one language is to be used to call code written in a second language, is it not preferable for the calling language to be the broader, general-purpose language, and the called code to be written in the special-purpose language? The two options are illustrated in Figures 1(a) and 1(b). Again, Bentley's paper shed light on the subject, in that he agrees that performance and logic suggest that flow-control elements, such as loops, are better implemented in general-purpose languages like Pascal, rather than in little languages. From a system design standpoint, we believe that the system illustrated in Figure 1(b) is the better option.

This analysis pushed us toward using a subroutine library packaged up in an API. Users can download the library, then call the functions contained in it from within his program. Inputs and outputs to an individual optimization model are variables within the main program, and can therefore be manipulated directly using the full facilities of the programming language.

An existing metamodeling system that we analyzed in this context was Matlab, a programming language that "provides core mathematics and advanced graphical tools for data analysis, visualization, and algorithm and application development" (http://www.Mathworks.com/). One of the available toolboxes for Matlab is the Optimization Toolbox, which enables general and large-scale optimization, including linear programming, quadratic programming, nonlinear least-squares, and nonlinear equations. Accordingly, Matlab could be viewed as an example of a language that provides access to optimization functionality via a subroutine library.

However, the evolution of Matlab showed us that it is more like a little language that has been expanded to include additional functionality, instead of an efficient general-purpose programming language with subroutine library access to special functionality. This shows in the performance of the Matlab language, especially in its slow handling of loops. Since we wanted to take advantage of good performance and flexibility in our master language, we shied away from Matlab's solution.

Several candidates emerged for the programming language of our library:

- Microsoft Visual Basic is easy to learn, allows for the fast creation of attractive user interfaces for programs, and has mechanisms

(a)

```
steel.mod:
set PROD;                # products
param rate {PROD} > 0; # tons produced per hour
param avail >= 0;        # hours available in week
param profit {PROD};    # profit per ton
param market {PROD};    # limit on tons sold in week
var Make {p in PROD} >= 0, <= market[p]; #tons produced
maximize Total_Profit: sum {p in PROD} profit[p] * Make[p];
                     # Objective: total profits from all products
subject to Time: sum {p in PROD} (1/rate[p]) * Make[p] <= avail;
                     # Constraint: total of hours used by all
                     # products may not exceed available hours
```

(b)

```
steel.dat:
set PROD := bands coils;
param:     rate  profit  market  :=
  bands    200   25       6000
  coils    140   30       4000
param avail := 40;
```

Example 1: *Input files.*

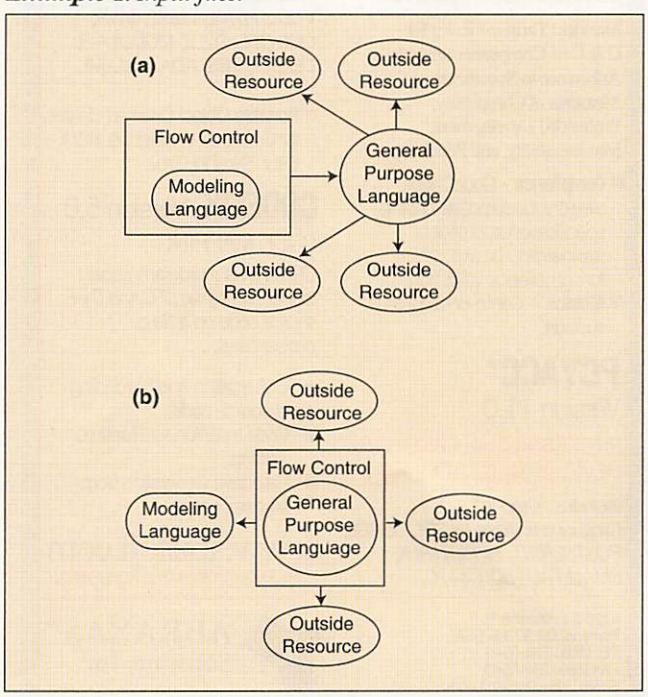
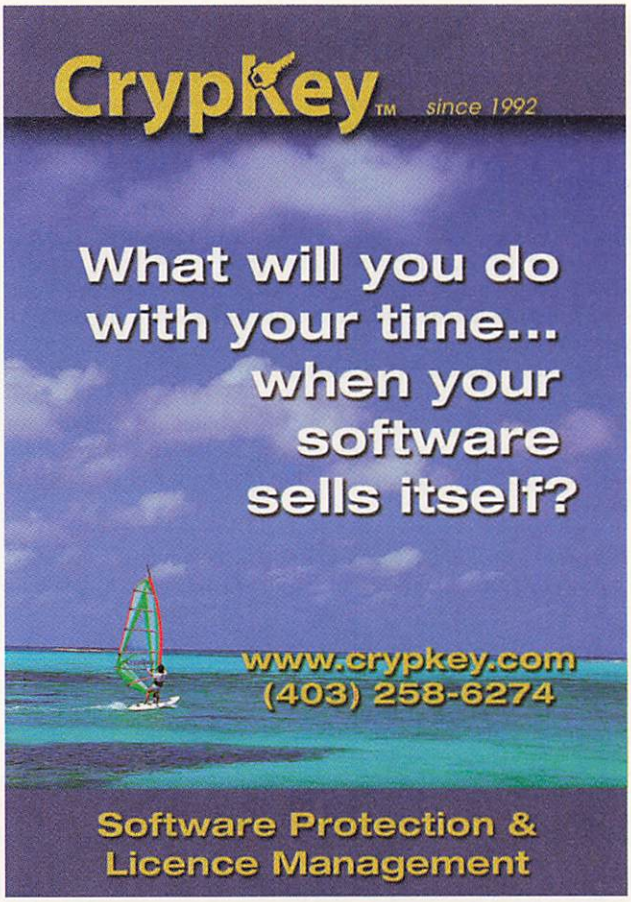


Figure 1: *Language design options.*


CrypKey™ since 1992
What will you do with your time... when your software sells itself?
www.crypkey.com
(403) 258-6274
Software Protection & Licence Management

Leader in language analysis tools since 1984!

CODECHECK®
Version 11.0
SOURCE CODE ANALYST

Includes "Drop-in" Rules for C & C++ Compliance analysis, Adherence to Specification, Measures of Complexity, Embedded Development, Maintainability, and Portability.

- **Compliance** – CodeCheck allows your corporate coding specification and project standards to be automated for compliance validation.
- **Metrics** – Common algorithm support.

PASCAL, PROLOG, FORTRAN, COBOL, BASIC, SGML, ASN, RPG, REXX, PL1, SNA, RTF, VISUAL BASIC, SQL2, DB2, VHDL, HTML, VMRL, JAVA, ODL-OQL, SQL3, MODULA-3, DELPHI, VBS, ADA, and XML.

- Portable Object Oriented classes for C, C#, C++, Delphi & JAVA – Error, Symbol, Tree.

PCYACC®
Version 11.0
PROFESSIONAL LANGUAGE DEVELOPMENT TOOLKIT

Includes "Drop-in" Language engines for SQL, dBASE, POSTSCRIPT, HYPERTALK, SMALLTALK-80, C++, C,

CODEFIX® Version 5.0
CODE MAPPING

CODEFIX is a programmable tool for modifying all C and C++ source code on a file or project basis.

- Obfuscation and Shrouding of source code.
- Code Insertion for Runtime Testing.
- Database Generation from source code.

www.abxsoft.com
Free Shipping anywhere in the world!

4726 S.E. Division St.
Portland, OR 97206 USA
TEL (503) 232-0540
FAX (503) 232-0543
E-mail: sales@abxsoft.com

ABRAXAS™ Software, Inc.

Linux $495 - Win2K/XP $995 - Unix-HP/AIX/Solaris $1995

to interact with Excel and Access. However, it has inconsistent syntax, making programming frustrating at times. Programs written in Visual Basic run slowly, and in our experience, they run unreliably, letting the same program behave differently when run repeatedly under the same conditions. This is unacceptable for many programming situations. Additionally, Visual Basic is nonportable, limiting users to Windows, and it is licensed, requiring a license for each programmer.

- C/C++ was an automatic candidate, given how widely the language is used for software development. A C/C++ compiler is available on almost every platform. Programs written in C/C++ tend to run fast (unless written poorly), and there are many existing subroutine libraries available to provide programmers with additional resources. The downside of C/C++ is that programming in the language is difficult and error prone, due to the way low-level memory management and addressing are left in the hands of the programmer. Also, while there are compilers on many platforms, code usually requires some customization for each platform.
- Java was a serious contender, with many strengths. The programming language is rigorously designed for large, object-oriented projects. Programs written in Java tend to be resistant to memory errors, thanks to an effective memory-management model that includes garbage collection. There are many existing Java subroutine libraries available to provide the programmer with additional resources. Finally, portability was a key goal in Java's design, and as a result, Java programs tend to run across several platforms. Java's downside relates to one of its upsides: The language's strict reliance on the object-oriented model makes learning the language difficult and writing small programs cumbersome.
- Perl has very strong built-in data parsing and reformatting functionality. Programs written in Perl are portable, running on many platforms, including all major UNIX variants and Windows versions. The language allows for easy procedural programming for smaller tasks, as well as object-oriented programming for larger projects. Also, there is a wide range of add-ons already available for Perl. This was important because it told us that programmers would have many resources available to them, and that creating such an add-on ourselves was likely to be easier and well documented. On the downside, for very large projects, Perl's object-oriented model is less rigorous than that of Java. Also, graphical user interfaces are harder to create in Perl than in Visual Basic or in some C/C++ toolkits.

After some consideration, Perl emerged as the winner. Visual Basic's design weaknesses and per-programmer licensing excluded it easily. C/C++ is too cumbersome. Java was a tempting option, but the ease with which researchers can throw together Perl programs made Perl our choice.

We implemented a Perl-AMPL library and packaged it as a module that is to be distributed via the Comprehensive Perl Archive Network (CPAN; http://www.cpan.org/). CPAN, a large collection of software and documentation that add extra functionality to Perl, is complemented by the Perl Automatic Upload Server (PAUSE). This server lets creators of Perl modules upload to CPAN, where their modules become available to Perl users for free. Because PAUSE makes it possible for Perl programmers around the world to contribute to CPAN, the archive contains thousands of Perl modules on a broad range of subjects: accessing different kinds of databases, communicating via different network protocols, interoperating with different data formats, and so on.

We need to mention here that there are some commercial products that fit the main requirements of the metamodeling system-design framework we have outlined in this article. For example, ILOG's OPL Studio (http://www.ilog.com/products/oplstudio/) includes the ability to build optimization models that are then

accessed from a subroutine library using Visual Basic, Java, or C/C++. The main program can generate data for the model dynamically, and the program can trigger the solving of the model and the generated data repeatedly. This indicates that OPL Studio includes the kind of functionality that we endorse for optimization metamodeling; namely, allowing the use of a general-purpose programming language that can make optimization calls to a subroutine library. However, we believe that the existence of CPAN is an important reason for selecting Perl as the general-purpose programming language of the system. By creating a framework that lets a large number of programmers contribute code that they perceive as helpful, CPAN has created the open-source framework by which a product (in this case, Perl) can outgrow the imagination of its own original creators.

The PerlAmpl Module

The PerlAmpl module was designed as an object-oriented Perl class. The *AMPL* object is an abstraction of an AMPL problem. The member functions let you set the problem's model, data, and options, after which member functions can be used to trigger the solver and access the results.

To use the API, include the module in your program with *use Math::Ampl;*, then initialize the library:

```
Math::Ampl::Initialize(inAmplDir, inAmplBin,
                                    inTempDir, [inPreserveFiles]);
```

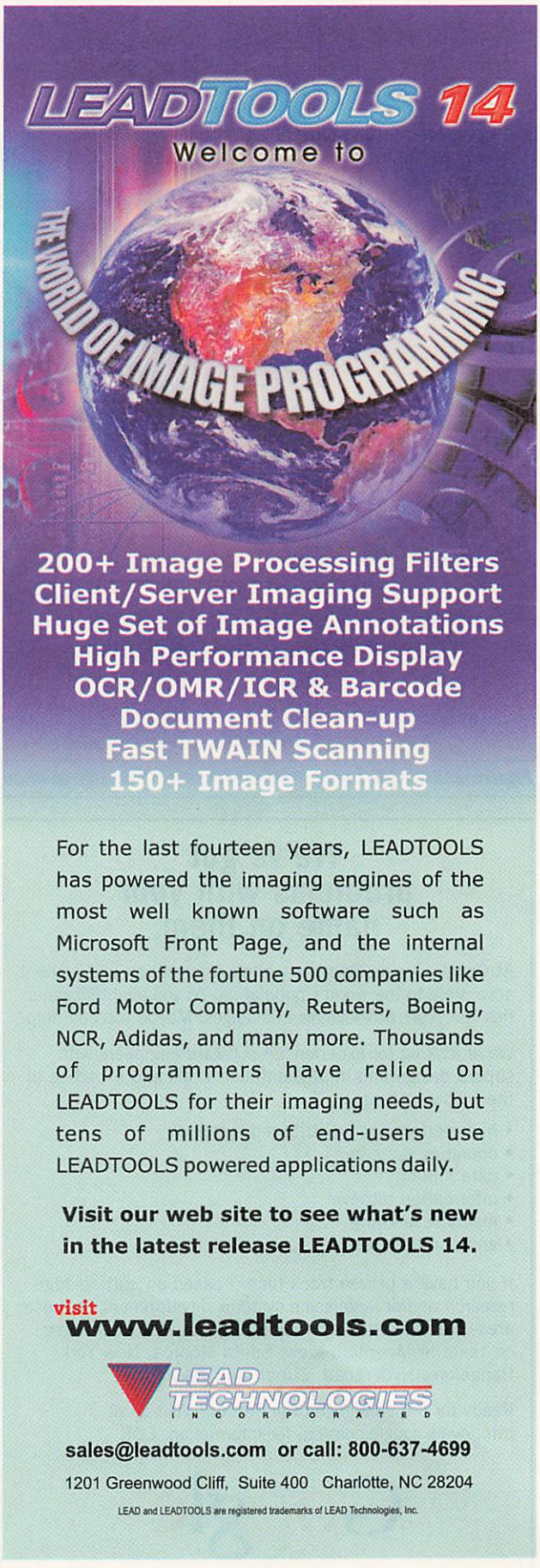
The argument *inAmplDir* is the directory containing the AMPL binary, *inAmplBin* is the name of the binary in that directory, *inTempDir* is a place to store temporary files, and *inPreserveFiles* is an optional argument (if 1, temp files are not deleted).

You next construct an instance:

```
$problem = new Math::Ampl;
```

Then, by using *$problem->Function(Arguments)* syntax, you can use the following instance member functions:

- *Input_Mod_Append(inModText)* to append *inModText* to the *mod* data, which will be passed to AMPL as the model.
- *Input_Mod_Clear* to clear the mod data.
- *Input_Dat_Add_Set_Item(inSet, inDimension1, ...)* adds a set item to the *dat data*. *inSet* is the name of the set. *inDimension1* is the first dimension of the item, and so on.
- *Input_Dat_Add_Param_Item(inParam, inDimension1, ..., inValue)* to add a *param* item to the *dat* data. *inParam* is the name of the *param*. *inDimension1* is the first dimension of the item, and so on. *inValue* is the value.
- *Input_Dat_Clear* to clear the *dat* data.
- *Input_Display_Add(inDisplayItem)* to add an item to be retrieved from AMPL using *Display*. After solving, the values of all added items can be retrieved in Perl using *Output_Display_Get_Text* or *Output_Display_Get_Value*.
- *Input_Display_Clear* to clear the list of *Display* items to retrieve.
- *Input_Option_Add(inOption, inValue)* to add an option to be set during the *Solve* command, right before AMPL actually runs the solver. *inOption* is the option to set, *inValue* is the value.
- *Input_Option_Clear* to clear the options to set during *Solve*.
- *Solve* to run AMPL on the problem. It returns -1 if an error occurred, which prevented the solver from being called, and 0 if the solver was called but could not generate a solution. It returns 1 if the solver found a solution.
- *Solve_Best_Solver(inMinOrMax, inObjective, inSolver1, ...)* to run *Solve* for each *inSolver*, keeping track of the best results. *inMinOrMax* must be either "min" or "max." The function uses *inMinOrMax* to decide if one result is better than the next. After trying all the solvers, *Solve_Best_Solver* reruns the best solver to fill the *Display* variables with the best values (unless the best solver was already the last one, in which case, it does


LEADTOOLS 14
Welcome to
THE WORLD OF IMAGE PROGRAMMING
200+ Image Processing Filters
Client/Server Imaging Support
Huge Set of Image Annotations
High Performance Display
OCR/OMR/ICR & Barcode
Document Clean-up
Fast TWAIN Scanning
150+ Image Formats
For the last fourteen years, LEADTOOLS has powered the imaging engines of the most well known software such as Microsoft Front Page, and the internal systems of the fortune 500 companies like Ford Motor Company, Reuters, Boeing, NCR, Adidas, and many more. Thousands of programmers have relied on LEADTOOLS for their imaging needs, but tens of millions of end-users use LEADTOOLS powered applications daily.
Visit our web site to see what's new in the latest release LEADTOOLS 14.
visit
www.leadtools.com
LEAD TECHNOLOGIES INCORPORATED
sales@leadtools.com or call: 800-637-4699
1201 Greenwood Cliff, Suite 400 Charlotte, NC 28204
LEAD and LEADTOOLS are registered trademarks of LEAD Technologies, Inc.

not need to be rerun). This function has been particularly useful for solving difficult nonlinear problems where the solution found may vary from solver to solver.

- *Output_Display_Get_Value(inParamOrVar, inDimension1, ...)* to get the retrieved value of a variable or parameter. The *inParamOrVar* must have been added using *Input_Display_Add* before calling *Solve*.
- *Output_Display_Get_Text(inParamOrVar)* to get the complete text block returned by AMPL for *inParamOrVar*. The *inParamOrVar* must have been added using *Input_Display_Add* before calling *Solve*.

The PerlAmpl module addresses many of the metamodeling issues mentioned here. The Perl environment facilitates the efficient handling of data and intermediate results, allows leveraging a wide variety of modules to maximize code reuse and minimize product development time, and makes formatting end results easy. We illustrate the system's capabilities with several examples.

Example: Handling Input

While the World Wide Web contains an enormous amount of data, it is frequently difficult to download them in the appropriate format. Perl modules, such as Finance::Quote, can be used

Data file:

U.S. TREASURY YIELD CURVE SUN, 4 JUL 1999, 11:32AM EDT

Bills	Mat Date	Current Price/Yield	Previous Price/Yield	Yld Chg	Prc Chg
3month	9/30/99	4.54(4.67)	4.54(4.67)	0.00	+0
6month	12/30/99	4.77(4.97)	4.77(4.97)	0.00	+0
1year	6/22/00	4.80(5.06)	4.80(5.06)	0.00	-0

Notes/ Bonds	Coupon	Current Mat Date	Previous Price/Yield	Price/Yield	Yld Chg	Prc Chg
2year	5.750	6/30/01	100-10+(5.57)	100-10+(5.57)	0.00	-0-00
5year	5.250	5/15/04	98-04(5.70)	98-04(5.70)	0.00	-0-00
10year	5.500	5/15/09	97-17(5.83)	97-17(5.83)	0.00	-0-00
30year	5.250	2/15/29	89-17+(6.01)	89-17+(6.01)	0.00	-0-00

[U.S. TREASURY YIELD CURVE]

Inflation Indexed Treasury	Coupon	Current Mat Date	Previous Price/Yield	Price/Yield	Yld Chg	Prc Chg
5year	3.625	7/15/02	00 (.00)	99-06(3.91)	-3.91	-99-06
10year	3.875	1/15/09	00 (.00)	99-07+(3.97)	-3.97	-99-08
30year	3.875	4/15/29	00 (.00)	99-09+(3.91)	-3.91	-99-10

B(c) Copyright 1999, Bloomberg L.P. All Rights Reserved. B(c) Copyright 1999 USA TODAY, a division of Gannett Co. Inc.

Figure 2: *Example use of PerlAmpl parsing Treasury yield curve data.*

We've got problems with your name on them.

At Google, we process the world's information and make it accessible to the world's population. As you might imagine, this task poses considerable challenges. Maybe you can help.

We're looking for experienced software engineers with superb design and implementation skills and expertise in the following areas:

- high-performance distributed systems
- operating systems
- data mining
- information retrieval
- machine learning
- and/or related areas

If you have a proven track record based on cutting-edge research and/or large-scale systems development in these areas, we have brain-bursting projects with your name on them in Mountain View, Santa Monica, New York, Bangalore, Hyderabad, Zurich and Tokyo.

Ready for the challenge of a lifetime? Visit us at http://www.google.com/ddj for information. EOE

Google™

C++ toolkits™

Just the Facts.

Standards Toolkit

(Bundle 1 of 6)

Features

- Robust implementation of the ANSI/ISO C++ Standard Template Library specification
- Multiple, extensive sets of collection classes

Benefits

- Extensible interface in the C++ Standard Library
- Flexibility to select the classes that best meet specific technical needs

Download a free evaluation of our C++ Toolkits.

Visit our website or contact one of our engineers:

Phone: 800.727.8674 Email: engineer@recursionsw.com.

RECURSION SOFTWARE, Inc. Software for Software Developers

www.recursionsw.com/cplus.htm

directly to download stock price data from web sites, such as Yahoo Finance. Perl modules also exist for accessing spreadsheets and databases. For example, DBD::Excel lets you access the data within an Excel file from a database interface, while Spreadsheet::TieExcel lets you tie an array within a Perl program to an Excel spreadsheet.

When a module is not available from CPAN, Perl's strong regular expression engine makes parsing unformatted (or inconveniently formatted data) an easy task. Figure 2 and Listing One present an example use of PerlAmpl parsing Treasury yield curve data that was then used in a computational study of credit risky bonds. The subroutine *read_file* reads in data on Treasuries downloaded from http://www.bloomberg.com/ (see Figure 2) and stores the yields and maturities of the 3-month, 6-month, and 1-year bills in Perl lists *@gYieldList* and *@gPeriodList*, respectively (it ignores the other information).

Example: Handling Intermediate Results and Formatting Output

We now show an example from finance — computing and plotting the mean-variance efficient frontier of a portfolio of three stocks — to illustrate how you could use the PerlAmpl module to run optimization problems multiple times, perform statistical analysis of data, plot graphs, and format output. To generate the efficient frontier, we need to solve the optimization problem in Example 2, where n is the number of stocks in the portfolio, x_i are decision variables corresponding to the allocation in each of the n stocks, r_{target} is the target portfolio return, and $E[r_i]$ and σ_{ij} are the expected values and the covariances of asset returns i, j, $i=1,\ldots,n$, $j=1,\ldots,n$, respectively. σ_{ij} equals the variance of return i.

Lists of expected returns, target returns, and a covariance matrix are passed to the optimization problem formulation using the PerlAmpl module. The forecasts for expected returns could be generated in Perl, for example, by running regressions or using time series techniques. A number of statistics Perl modules, such as Statistics::Regression, available free from http://www.cpan.org/, can help with the statistical analysis of data. Also, Perl modules, such as Math::Matlab, provide additional capabilities by allowing for calling outside statistical software and collecting the results.

The portfolio optimization problem is nonlinear, so the FSQP solver is called from within AMPL. The results are stored in a hash in Perl. JGraph (a free graphing software project under Linux; http://sourceforge.net/projects/jgraph/) plots the results dynamically. The advantage of this type of organization is that the whole metamodel can be run efficiently from beginning to end while keeping the same output format. Output formats that are frequently convenient, for example, are Latex table, Excel, or HTML. Listing Two contains a subroutine for printing the optimal standard deviation for each level of target portfolio return directly in a Latex table format.

The output of this program is a file efficientFrontier.jgr and this table:

```
Expected Return & Standard Deviation \\
0.080 & 0.105 \\
0.085 & 0.105 \\
0.090 & 0.105 \\
0.095 & 0.107 \\
0.100 & 0.118 \\
0.105 & 0.136 \\
0.110 & 0.160 \\
0.115 & 0.186 \\
0.120 & 0.214 \\
```

The efficientFrontier.jgr file contains the text in Example 3. Using the command *jgraph efficientFrontier.jgr > efficientFrontier.ps*, the file can be converted to a picture (Figure 3), which can then be used, for example, in a Latex file. The conversion of the Jgraph file to a PostScript file can be automated by including it directly in the Perl program.

Example: Handling Intermediate Results and Formatting Output

We successfully used the PerlAmpl module in computational comparisons of the efficiency of multiperiod versus single-period portfolio optimization techniques. The module can be similarly used in other optimization metamodels that involve simulations at multiple stages.

Consider an example of a multiperiod optimization problem: Given a portfolio of stocks and information about their future expected values at multiple times in the future, solve two different portfolio optimization problem formulations to determine optimal portfolio allocations for each time period. Then run simulations to test which of the two allocations results in a better final period portfolio return. To compare the two optimization formulations, you would need to compare cumulative returns at the end of the time horizon. At every point in time, you need to keep track of the portfolio value, rebalance the portfolio, simulate returns for one period ahead, and recompute the value of the portfolio.

AMPL contains several built-in random generator functions, such as *Beta(a,b)*, *Cauchy()*, *Exponential()*, *Normal(mean, standard deviation)*, *Poisson(parameter)*, and *Uniform(lower limit, upper limit)*. All of the previous random distributions can

$$\text{minimize}\left\{\sum_{i=1}^{n}\sum_{j=1}^{n}\sigma_{ij}x_i x_j \,\middle|\, \sum_{i=1}^{n}E[r_i]x_i \geq r_{\text{target}}\right\}$$

Example 2: *Optimization problem.*

```
newgraph
 xaxis
  label : Standard Deviation
 yaxis
  label : Expected Portfolio Return
newcurve marktype none linetype solid linethickness 1
     label : Efficient Frontier
         pts  0.105347520141672 0.08  0.105347520141672
     0.085  0.105347520141672 0.09 0.107279541386044
     0.095  0.118133822421862 0.1  0.136400513195515
     0.105 0.159553752697954 0.11  0.185775671173596
     0.115  0.213940645974532 0.12
legend defaults
 x 0.17 y 0.10
```

Example 3: *Text contained in the file efficientFrontier.jgr.*

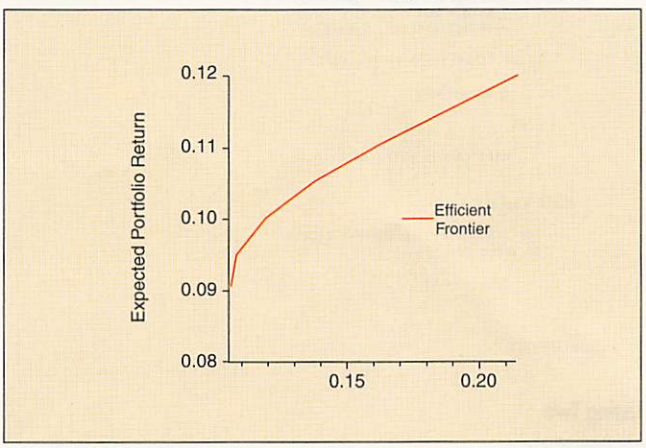


Figure 3: *Converting file to a picture.*

also be generated in Perl by using, for example, the Math::Cephes Perl module, which contains over 150 mathematical functions. In some contexts, such as the multiperiod portfolio optimization example we mention, it is significantly more convenient to run the simulations within the main Perl program because of the need to process intermediate optimization results. Moreover, Perl offers a wider variety of random generator functions. Listing Three is Perl code for a simplified simulation example in which the asset returns in a portfolio are drawn from a multivariate normal distribution with prespecified expected values vector and covariance matrix. Currently, there is no provision for generating correlated random variables from within AMPL. We use the Perl library random_multivariate_normal.pl (http://www.cpan.org/).

Conclusion

While the PerlAmpl module is effective and useful in its current state of development, possible enhancements include parallelizing the execution of solvers to take advantage of multiprocessors, as well as networking the PerlAmpl module to allow client computers to solve optimization metamodels by leveraging software on server computers.

DDJ

Listing One

```
%kMonthHash = ("Jan" => 1, "Feb" => 2, "Mar" => 3, "Apr" => 4,  "May" => 5,
               "Jun" => 6, "Jul" => 7, "Aug" => 8, "Sep" => 9, "Oct" => 10,
               "Nov" => 11, "Dec" => 12);
%kConvertHash = ("3month" => 90, "6month" => 180, "1year" => 360,
               "2year" => 720, "5year" => 1800, "10year" => 3600,
               "30year" => 10800);
sub read_file
{
    my($inFile) = @_;
    my($state, $temp, $period, $yield);
    open(INPUT, $inFile) || die("Cannot open $inFile");
    $state = 0;
    while (<INPUT>)
    {
        if ($state == 0)
        {
            #if (/U.S. Treasury yield curve/)
            if (/\*U.S. Treasuries\*/)
            {
                $state = 1;
            }
        }
        elsif ($state == 1)
        {
            #READ IN CURRENT DATE
            # Sun, 4 Jul 1999, 11:32am EDT
            if (/[A-Z][a-z][a-z],\s*([0-9]+)\s*([A-Z][a-z][a-z])\s*([0-9]+),/)
            {
              $gDay = $1;
              #assign a numerical value to the month according to %kMonthHash
              $gMonth = $kMonthHash{$2};
              $gYear = $3;
              $state = 2;
            }
            else
            {
              die("cannot parse date");
            }
        }
        elsif ($state == 2)
        {
            #if (/Bills/)
            if (/Prc Chg/)
            {
                $state = 3;
            }
        }
        elsif (($state == 3) || ($state == 5))
        {
            if ((/Notes/) || (/\[U.S. Treasury Yield Curve\]/))
            {
                $state++;
            }
            elsif (/^(\S+)\s+[^(]+\(([^)]+)\)/) #READ IN RELEVANT DATA
            {
                $period = $kConvertHash{$1};
                push(@gPeriodList, $period);
                $yield = $2;
                push(@gYieldList, $yield);
            }
            elsif (/\s*[+-][0-9+-]+\s*$/)
            {
                # do nothing
            }
            else
            {
                die("cannot parse data: $_");
            }
        }
        elsif ($state == 4)
        {
            #if (/Bonds\s+Coupon\s+Mat\s+Date/)
            if (/Prc Chg/)
            {
                $state = 5;
            }
        }
    }
    close(INPUT);
}
```

Listing Two

```
use Math::Ampl;
use strict;

#SPECIFICATION OF INPUT DATA:
my($NumStocks) = 3;
#vector of expected returns
my(@ExpectedReturnsList) = (0.08, 0.09, 0.12);
#vector of standard deviations for each stock
my(@StdDeviationsList) = (0.15, 0.20, 0.22);
#vector of target portfolio returns
my(@TargetReturnsList) = (0.08, 0.085, 0.09, 0.095, 0.10, 0.105,
                                                    0.11, 0.115, 0.12);
#RESULTS STORAGE:
#hash table to store optimal portfolio standard deviation results after
#solving all optimization problems
my(%OptimalStdDeviationsHash) = ();
#hash table to store optimal portfolio holdings after solving the
#portfolio optimization problem for each value of TargetReturn
my(%OptimalHoldingsHash) = ();

#OUTPUT FILES:
my($kOutputFile) = "efficientFrontier.out"; #to store Latex table
my($kFrontierGraphFile) = "efficientFrontier.jgr"; #to store graph

#DECLARATION OF AMPL PROBLEM INSTANCE:
my($gProblem);

#optimization model problem to be solved: minimize portfolio variance
#subject to constraints on portfolio expected return

#optimization model file to be passed to AMPL using PerlAmpl

sub setup_ampl_mod
{
    my($mod);
    $mod =<<EODATA;

    param NumStocks;
    param ExpectedReturns{1..NumStocks};
    param StdDeviations{1..NumStocks};
    param TargetReturn;

    var holdings{1..NumStocks};

    minimize portfolio_variance:
    sum{i in 1..NumStocks}
            (StdDeviations[i]*StdDeviations[i]*holdings[i]*holdings[i]);

    subject to portfolio_target_return:
    sum{i in 1..NumStocks} (ExpectedReturns[i]*holdings[i]) >= TargetReturn;

    subject to portfolio_total:
    sum{i in 1..NumStocks}(holdings[i]) = 1;
EODATA
        $gProblem->Input_Mod_Append($mod);
}
#optimization data file to be passed to AMPL using PerlAmpl
sub setup_ampl_dat
{
    #target portfolio return passed for this instance of the problem
    my($inTargetReturn) = @_;
    my($iStock);

    $gProblem->Input_Dat_Add_Param_Item("NumStocks", $NumStocks);
    $gProblem->Input_Dat_Add_Param_Item("TargetReturn", $inTargetReturn);
    for($iStock = 1; $iStock <= $NumStocks; $iStock++)
    {
        $gProblem->Input_Dat_Add_Param_Item("ExpectedReturns",
                                        $iStock,$ExpectedReturnsList[$iStock-1]);
        $gProblem->Input_Dat_Add_Param_Item("StdDeviations",
                                        $iStock,$StdDeviationsList[$iStock-1]);
    }
}
#request for AMPL to keep track of variables of interest

sub setup_ampl_display
{
    $gProblem->Input_Display_Clear;
    $gProblem->Input_Display_Add("solve_result");
    $gProblem->Input_Display_Add("portfolio_variance");
    $gProblem->Input_Display_Add("holdings");
}
#script for running the problem in AMPL and obtaining the results
sub solve_problem
{
    #target portfolio return passed for this instance of the problem
    my($inTargetReturn) = @_;
    my($solved);
```

(continued on page 24)

Modeling Made Easy!

CA's AllFusion Modeling Suite simplifies and accelerates the complex aspects of analyzing, designing and implementing applications and business processes!

AllFusion® Modeling Suite BUNDLE

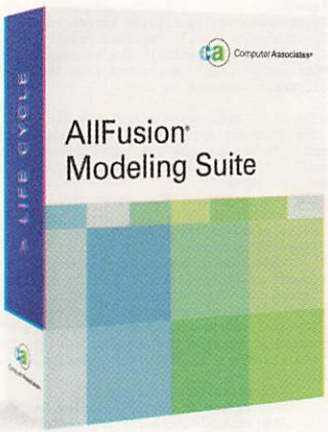
AllFusion® Modeling Suite is made up of the industry's most powerful solutions. Use them individually or together for unsurpassed power to share models, design high-quality business applications, verify the integrity of your database applications, automate capturing business and IT processes, as well as work with complex metadata independent from an ERP provider.

SAVE $$$

- AllFusion® ERwin® Data Modeler
- AllFusion® Model Manager
- AllFusion® Process Modeler
- AllFusion® Component Modeler
- AllFusion® Data Model Validator

P26 0209* **$7,672.99**

FREE 15-Day trial of AllFusion® solutions at programmersparadise.com/ca



AllFusion ERwin Data Modeler

Industry-Leading Data Modeling and Database Design

- Quickly create high-quality databases
- Separated logical and physical models—share and reuse models
- Complete/compare technology
- Integrates with **AllFusion® Saphir Option**

P26 010H* **$3,845.99**

Award-Winning Data Modeling Products

Programmer's Paradise 2003 Top Product

JDJ Readers' Choice Award — intelligent enterprise — intelligent enterprise

ACCESS VB·SQL ADVISOR MAGAZINE — Companies to Watch

AllFusion Model Manager

Enterprise Model Management Solution

- Synchronize and share metadata between data and process models
- Change control with security features

P26 0139* **$1,518.99**

AllFusion Process Modeler

Business Process Modeling

- Supports powerful, proven IDEFO, IDEF3 and DFD modeling techniques
- Enables powerful report writing with Report Template Builder

P26 0104* **$2,586.99**

AllFusion Component Modeler

Component Modeling

- Provides full, easy support for UML 1.4
- Helps ensure effortless information porting via XML and XSL
- Enjoy advanced model-to-model and model-to-code synchronization

P26 024C* **$1,914.99**

AllFusion Data Model Validator

Database Validation

- Easily verifies the integrity of database applications in production
- Incorporates changes into an existing database simply
- Comprehensively analyzes data structures, keys, indexes and more

P26 010V* **$1,898.99**

** Includes 1 year maintenance*

De-mystify Modeling!

Order any or all of CA's Award-winning Modeling Solutions from Programmer's Paradise today!

800-445-7899 ext. 7248

Computer Associates®

© 2004 Computer Associates International, Inc. (CA). All trademarks, trade names, service marks and logos referenced herein belong to their respective companies.

Programmer's Paradise®

(continued from page 22)

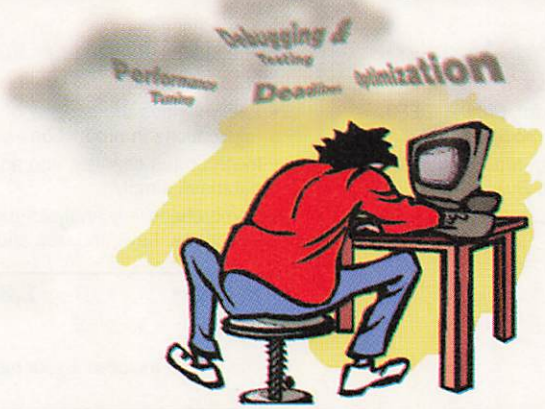
```
    $gProblem->Input_Mod_Clear;
    $gProblem->Input_Dat_Clear;
    setup_ampl_dat($inTargetReturn);
    setup_ampl_mod();
    setup_ampl_display();
    $solved = $gProblem->Solve;
    return $solved;
}
#PRINT GRAPH OF EFFICIENT FRONTIER DYNAMICALLY
#the output is a .jgr file which can then be converted to a .ps file

sub print_graph
{
    my($graph);
    my($targetReturn); #portfolio target return, read from list
    my($portfolioVariance); #optimal result from optimization problem
    my($portfolioStdDeviation); #to be computed from portfolio variance
    my(@CurrentHoldings); #obtained from AMPL output
    my($iStock); #counter

    open(GRAPH, ">$kFrontierGraphFile") ||
        die ("Cannot open file \"$kFrontierGraphFile\" for graph: $!");
    $graph = "newcurve marktype none linetype solid linethickness 1";
    $graph .= " label : Efficient Frontier\n";
    $graph .= "\tpts ";
    foreach $targetReturn (@TargetReturnsList)
    {
        solve_problem($targetReturn);
        $portfolioVariance =
            $gProblem-> Output_Display_Get_Value("portfolio_variance");
        $portfolioStdDeviation = sqrt($portfolioVariance);
        $graph .= " $portfolioStdDeviation $targetReturn ";
#store optimal standard deviations if necessary
        $OptimalStdDeviationsHash{"$targetReturn"} = $portfolioStdDeviation;
#if necessary, store also the optimal holdings for each value of TargetReturn
        for($iStock = 1; $iStock <= $NumStocks; $iStock++)
        {
            $CurrentHoldings[$iStock-1] =
                $gProblem->Output_Display_Get_Value("holdings",$iStock);
            $OptimalHoldingsHash{"$targetReturn,$iStock"} =
                $CurrentHoldings[$iStock-1];
        }
    }
    print GRAPH "newgraph\n xaxis\n label : Standard Deviation\n";
    print GRAPH "yaxis \n ";
    print GRAPH "label : Expected Portfolio Return\n";
    print GRAPH "$graph\n\n";
    print GRAPH "legend defaults\n x 0.17 y 0.10\n";
    close GRAPH;
}
#PRINT A TABLE WITH RESULTS IN LATEX TABLE FORMAT
sub print_table
{
    my($portfolioStdDeviation);
    my($targetReturn);

    open(OUTPUT, ">$kOutputFile") ||
        die ("Cannot open file \"$kOutputFile\" with results: $!");
    printf OUTPUT "%s %s ", "Expected Return", "&";
    printf OUTPUT "%s %s \n", "Standard Deviation", "\\\\";

    foreach $targetReturn (@TargetReturnsList)
    {
        $portfolioStdDeviation = $OptimalStdDeviationsHash{$targetReturn};
        printf OUTPUT "%2.3f %s ", $targetReturn, "&";
        printf OUTPUT "%2.3f %s \n",  $portfolioStdDeviation, "\\\\";
    }
    close OUTPUT;
}
#MAIN
#initialization
Math::Ampl::Initialize($kAmplDir, $kAmplBin, $kTempDir, 1);
$gProblem =  new Math::Ampl;

print_graph();
print_table();
```

Listing Three

```
#Generates one path, and returns a List of Lists indexed by [time
#period, asset number]. The entries equal single-period returns for
#each stock. Single-period returns are multivariate normal random variables.

sub create_scenario
{
    #pass number of stocks in portfolio and number of time periods ahead
    my($inNumStocks, $inNumPeriods) = @_;

    my(@SimulatedReturnsList); #single time period simulated returns
    my(@ScenarioLoL); #list of lists of asset returns for each time period
    my($iT); #time period counter

    for ($iT = 0; $iT < $inNumPeriods; $iT++)
    {
        @SimulatedReturnsList =
                        random_multivariate_normal($inNumStocks,
                            @ExpectedReturnsList, @CovarianceMatrixList);
        #add simulated returns for time period iT to scenario path
        @ScenarioLoL[$iT] = [ @SimulatedReturnsList ];
    }
    return (@ScenarioLoL);
}
```

DDJ

Don't Let Deadlines Cloud Your Programming



Dr. Dobb's Journal CD-ROM - Release 15

This new release gives 24/7 access to source code, algorithms, programming paradigms, columns, and much more from 16 years — ***YES, 16 YEARS*** and over 190 issues of ***Dr. Dobb's Journal!***

No more taking time to search through issues of *DDJ* to find information. Just cut and paste right into your programming project. No bandwidth issues either! Information is just a click away on your CD Drive — 24/7!

Just Released and Now Available

www.ddj.com/cdrom/

ORDER TODAY!

Online: www.ddj.com/cdrom/
Telephone: 1.800.444.4881 (North America)
1.785.838.7500 (International)
1.785.838.7566 (Fax)

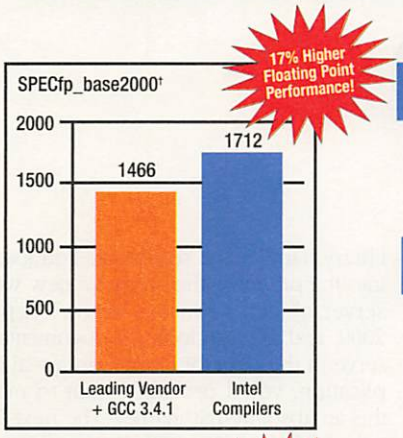
CMP

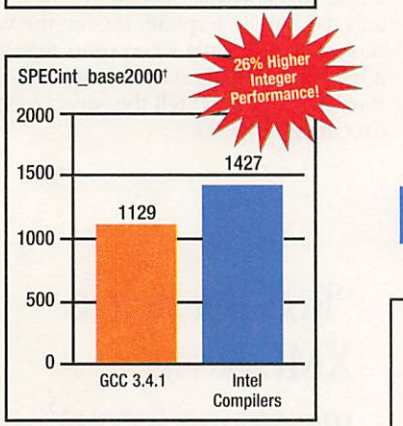
Dr. Dobb's CD-ROM LIBRARY

Maximize Your Application Performance

Intel® C++ and Visual Fortran 8.1

The latest version of Intel's fast compilers is here! Give your application a boost in performance with little or no source code modifications.





† • Intel® C++ Compiler 8.1, Extended Memory 64 Technology Edition for Linux*, Intel® Fortran Compiler 8.1, Extended Memory 64 Technology Edition for Linux*, GCC 3.4.1, PGI Fortran 5.2.1 x86-64
• Intel® Xeon™ Processor supporting Intel® EM 64T, 3.6GHz, 8GB Memory, 1MB L2 Cache
• RedHat* EL AS 3, TaroonUpdate 2, Kernel 2.4.21-12.EL, glibc 2.3.2

Performance

Outstanding performance on Intel architecture including Intel® Pentium® 4, Intel® Xeon™ and Intel® Itanium® 2 processors, now including support for Intel® Extended Memory 64 Technology.

Compatibility

Intel C++ for Windows →
- Integrates into Microsoft Visual Studio* .NET

Intel C++ for Linux →
- Source and object compatible with gcc version 3.4

Intel Visual Fortran for Windows →
- Strong compatibility with Compaq* Visual Fortran

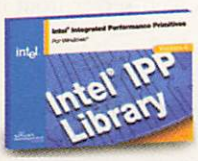
Intel Fortran for Linux →
- Compaq Visual Fortran front-end language features

Support

1 year of product upgrades and Intel Premier Support included

"We tried the Intel® C++ Compiler for Windows and...(it) generated impressive code, worked amazingly well with our Integrated Development Environment, and delivered substantial performance improvements...We recommend this to any C++ developer building applications for delivery on Windows-based systems using Intel processors."

—Mike Marchywka, Chief Scientist; EyeWonder, Inc.



Intel® Integrated Performance Primitives Library 4.1

Highly optimized code for graphics, multimedia, math and signal processing.

Intel® Thread Checker 2.1

Detects Win32 and OpenMP threading bugs, race conditions, deadlocks and more.

YOU SAVE UP TO $70!	Paradise #	Retail	Discount
Intel® C++ Compiler 8.1 for Windows	I23 0B8A	$399.00	$378.99
Intel® Visual Fortran 8.1 Standard	I23 0B8C	$499.00	$474.99
Intel® Visual Fortran 8.1 Professional	I23 0B8D	$1,399.00	$1,328.99
Intel® IPP Library 4.1 for Windows	I23 0B8E	$199.00	$189.99
Intel® Thread Checker 2.1 for Windows with VTune Analyzer 7.2	I23 0BDF	$1,198.00	$1,138.99

Programmer's Paradise®

To order or request additional information call:
800-423-9990
Email: intel@programmers.com

FREE trials available at:
programmersparadise.com/intel

© 2004 Intel Corporation Intel, the Intel logo, Pentium, Itanium, Intel Xeon and VTune are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. *Other names and brands may be claimed as the property of others.

RSS, Ruby, & the Web

Libraries that make RSS and web work easy

DAVE THOMAS

Really Simple Syndication (RSS) is the protocol of choice for disseminating news on the Internet—and with good reason. As a consumer of information, RSS feeds give you the convenience of consolidating all the data you receive into a single, easy-to-scan format. And if you're creating or publishing information, RSS is an easy-to-implement way of getting your data onto people's desktops.

Many people use RSS to read summaries of weblogs and other news sources. This is natural—these sites typically provide an RSS feed that can be read by RSS clients (known as "aggregators"). But RSS is far more than "Dear Diary, Yesterday I had pizza." With some simple tools, you can extract information from nonRSS sources, convert it to RSS, and summarize it alongside the other feeds in your aggregator. You can use RSS to collect and summarize information from your projects and from your life.

For example, in my publishing business, I work with a number of author teams, each using a part of our CVS repository. I use a trivial script to extract information every time someone commits a change to the repository, summarizing the change in an RSS feed. I'm also interested in the Amazon sales ranks of all our titles, so another simple script downloads the information and converts it into another RSS feed. I also extract and convert ToDo items from my calendar application. Throughout the day, my RSS aggregator notifies me when this information changes. It's like having a synoptic display of important information, tailored to your needs, all with very little work.

To make this happen, you need some tools. The recently released 1.8.2 version of Ruby (http://www.ruby-lang.org/) comes complete with a full set of libraries that make working with the Web and RSS easy. To introduce these, in this article, I show how to:

- Create a pure Ruby web server.
- Write a (very) basic weblog.
- Add an RSS feed to that weblog.
- Create RSS feeds from other data sources.

Finally, I look at the other side of the coin—taking an RSS feed and converting it into an HTML fragment suitable for embedding in a web page's sidebar.

Basic Blogging

At its simplest, a weblog (or blog) is a web page that displays a set of articles in reverse chronological order (newest at the top). When the writer adds a new story, it appears at the head of the list—the oldest story might drop off if the list reaches a given size.

There are many software packages and dedicated sites that implement fully fledged blogging solutions. However, as an introduction to both Ruby and its web libraries, I implement a very basic blog from scratch.

This blog runs in a web server. I could have written something that runs in Apache, but instead I make it standalone. Listing One is a complete web server that handles multiple mime types, directory indexing, CGI execution, user directories, partial content fetches, and more. Not bad for four lines of code. Of course, the WEBrick library handles the real work. You load it into the program, then create a new webserver object. I run our server on port 2000, and tell it to look for documents to serve in the directory html/ (in a real application, you'd probably want to make this an absolute pathname). The next line tells the Ruby interpreter to shut the web server down cleanly when users interrupt it by typing Ctrl-C on the console used to start it. Finally, you tell the server to start executing requests.

"RSS is a simple XML-based messaging format"

You can run this web server from the command line:

```
ruby webserver1.rb
```

Point a browser at http://localhost:2000/ and you should see a document served from the html/ directory.

Adding Weblog Capabilities

For illustrative purposes, I create a basic weblog that displays the documents in or below the web server's document root directory, showing the 10 most recent documents, newest first. I start by writing a class to represent a single article (Listing Two). The *initialize()* method is the class's constructor. It takes the name of a file containing the article and reads in the contents. It looks for a title, searching for either the contents of the HTML <*title*>

Dave is a principle in The Pragmatic Programmers and author of Programming Ruby. *He can be contacted at http://www.pragmaticprogrammer.com/.*

Programmer's Paradise®

Your best source for software development tools!

GUARANTEED BEST PRICES*

Should you see one of these products listed at a lower price in another ad in this magazine, CALL US! **We'll beat the price**, and still offer our same quality service and support!

*Terms of the offer:

- Offer good through Jan. 31, 2005
- Applicable to pricing on current versions of software listed
- January issue prices only
- Offer does not apply towards obvious errors in competitors' ads
- Subject to same terms and conditions

PROGRAMMER'S PARADISE GUARANTEED BEST PRICES

LEADTOOLS
Raster Imaging Pro for .NET
by LEAD Technologies

Extend and simplify the .NET framework and GDI+ with LEADTOOLS. Support for over 22 file formats including TIFF, JPG, J2K*, PDF*, and GIF using various compression schemes like JPEG, JPEG2000, LZW, CCITT G3/G4, and CMP. Scan, capture, 200+ image processing filters, high speed display (resize, rotate, quality controls), and much more. Lots of sample source code to get you started quickly!

*Requires plug-in

Paradise #
L05 053F
$829.99

www.programmersparadise.com/lead

dtSearch Web with Spider

Quickly publish a large amount of data to a Web site, with "blazing speeds" *(CRN Test Center)* searching.

- Features over a dozen indexed and fielded data search options.
- Highlights hits in XML, HTML and PDF, while displaying links and images; converts other files ("Office," ZIP, etc.) to HTML with highlighted hits.
- Spider adds a third-party site to a site's own searchable database.
- Optional API supports SQL, C++, Java, and all .NET languages.

Single Server
Paradise #
D29 0726
$888.99

Download dtSearch Desktop with Spider for immediate evaluation

"The most powerful document search tool on the market"—*Wired Magazine*

www.programmersparadise.com/dtsearch

c-tree Plus®
by FairCom

With unparalleled performance and sophistication, c-tree Plus gives developers absolute control over their data management needs. Commercial developers use c-tree Plus for a wide variety of embedded, vertical market, and enterprise-wide database applications. Use any one or a combination of our flexible APIs including low-level and ISAM C APIs, simplified C and C++ database APIs, SQL, ODBC, or JDBC. c-tree Plus can be used to develop single-user and multi-user non-server applications or client-side application for FairCom's robust database server—the c-tree Server. Windows to Mac to Unix all in one package.

NEW! SQL Support!

Paradise #
F01 0131
$850.99

www.programmersparadise.com/faircom

TX Text Control 11.0
Word Processing Components

NEW!

TX Text Control is royalty-free, robust and powerful word processing software in reusable component form.

- .NET WinForms control for VB.NET and C#
- ActiveX for VB6, Delphi, VBScript/HTML, ASP
- File formats RTF, DOC, HTML, XML
- PDF export without additional 3rd party tools or printer drivers
- Nested tables, headers & footers, hyperlinks, bullets, numbered lists, multiple undo/redo
- Ready-to-use toolbars and dialog boxes

Professional Edition
Paradise #
T79 021V
$739.99

Download a demo today.

www.programmersparadise.com/theimagingsource

Miscrosoft Visual Studio .NET Enterprise Developer

Visual Studio .NET 2003 provides developers with powerful tools and servers for rapidly building XML Web services and enterprise applications. Visual Studio .NET 2003 Enterprise Developer builds on the power of Visual Studio .NET 2003 Professional by including additional capabilities for enterprise development teams building mission-critical applications that target any device and integrate with any platform.

*As a free bonus for purchasing Visual Studio .Net Enterprise Developer 2003 Upgrade and while supplies last, receive a full licensed copy of IP*Works! Zip for .Net by /n software, a $245 value.*

Paradise #
M47 M27E
$951.99

www.programmersparadise.com/microsoft

Multi-Edit 9.10
by Multi-Edit Software Inc.

Multi-Edit 9.10 provides a single environment to control all your VCS programs and compilers, while integrating with existing RAD environments. Easily handle large DOS/Windows, UNIX, binary, and Macintosh files in over fifty programming languages immediately out of the box. Fully customizable, Multi-Edit's sophisticated search/replace feature now includes support for Perl 5 Regular Expressions. The Evolve XBase AddOn and Polystyle code formatter are integrated into every purchased copy of Multi-Edit, absolutely free! Seamlessly integrate with TopStyle, CSE Validator, HTML Tidy, Beyond Compare and other favorite accessories.

New Version! 9.10

Paradise #
A30 015Y
$208.99

www.programmersparadise.com/multiedit

IQstudio™
by Media Cybernetics

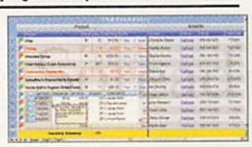
Develop sophisticated, Windows®-based image analysis applications with IQstudio.

- Easier testing & debugging
- Built-in Microsoft® Visual Basic® for Applications (VBA)
- Supports Visual Basic, C++, and managed .Net languages such as VB.NET and C#
- Objects for image processing, filtering, manual/automatic measurement, calibration, annotation, reporting, databasing, and camera acquisition (option)

Paradise #
CB8 0001
$3,389.99

www.programmersparadise.com/IQstudio

FarPoint's Spread
by FarPoint Technologies



When you need high-performance, fully-customizable grid/spreadsheet components for ASP.NET or Windows Forms development, join the professional developers who continually turn to the leader in grid/spreadsheet component technology. Spread for Web Forms and Spread for Windows promises to reduce your development time and costs by using our extensive programming models so that you can deliver to your customers a reliable feature rich user experience.

Spread for Web Forms
Paradise # F02 013E
$561.99

Spread for Windows Forms
Paradise # F02 013F
$749.99

www.programmersparadise.com/farpoint

Paradise Picks

DevTrack 5.6
Powerful Defect and Project Tracking
by TechExcel

DevTrack, the market-leading defect and project tracking solution, comprehensively manages and automates your software development processes. DevTrack 5.6 features sophisticated workflow and process automation, seamless source code control integration with VSS, Perforce and ClearCase, robust searching, and built-in reports and analysis. Intuitive administration and integration reduces the cost of deployment and maintenance.

programmersparadise.com/techexcel

Paradise #
T34 0201
$482.99

IP*Works! Red Carpet Subscriptions
by /n software

IP*Works! Red Carpet™ Subscriptions give you everything in one package: communications components for every major Internet protocol, SSL and SSH security, S/MIME encryption, Digital Certificates, Credit Card Processing, ZIP compression, Instant Messaging, and even e-business (EDI) transactions. .NET, Java, COM, C++, Delphi, everything is included, together with per developer licensing, free quarterly update CDs and free upgrades during the subscription term.

programmersparadise.com/nsoftware

Paradise #
D77 0148
$1264.99

800-445-7899 • programmersparadise.com

Prices subject to change. Not responsible for typographical errors.

(continued from page 26)
element or the first *<h1>* element. If neither is found, the title is set to the name of the file. Because the articles are regular HTML files, it also removes everything but the content of the *<body>* element. The resulting title, filename, and body are saved in instance variables (with names that start with @ signs). Make these accessible outside the class via the attributes you declare near the top of the class.

The method *Article.list* (Listing Three) returns an array of up to 10 *Article* objects, sorted in reverse chronological order. It illustrates some nice features of Ruby, including three uses of Ruby's blocks.

Ruby blocks are chunks of code (between *do/end* keywords or braces) that are associated with a method call. The method can invoke the block multiple times, and when the code in the block completes, control is returned to the associated method. The *list()* method uses three blocks. The first occurs on the call to *Dir.chdir()*. This method makes the given directory the process's current working directory and invokes the associated block (which, in this case, contains the rest of the body of the method). When the block finishes, control returns to the *chdir()* method, which resets the current directory back to the value it had previously.

Inside this block, *Dir.glob* returns an array of filenames that match a given pattern. The ** part of this pattern tells the *glob* to match in the current directory and in its subdirectories, and *.html matches filenames ending with "html."

The next line sorts this list of names according to the modification time of the associated file. This uses the new (in Ruby 1.8) *sort_by()* method that lets you sort a collection based on some value derived from the elements in that collection (in this case, the file's modification time).

Finally, the sorted list of names is reversed and up to 10 entries are extracted. This list is passed to the *map()* method, which creates a new array by passing each entry in turn to the given block and populating the result array with the value returned by the block. In this case, the *map()* construct converts the list of filenames into an array of *Article* objects.

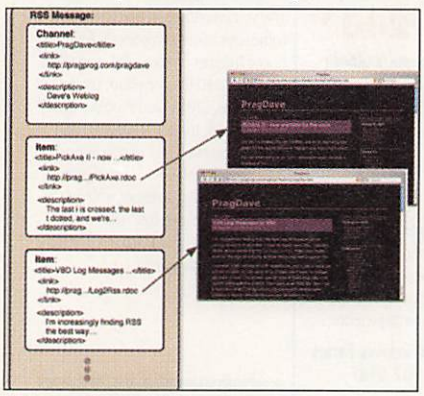


Figure 1: *An RSS message contains a channel header followed by zero or more items.*

It's interesting to compare the length of the method with the length of its description. Ruby code can be remarkably compact and yet (once you're familiar with the language) very readable.

With this infrastructure in place, you're ready to write a blog. You do this by adding a servlet to the existing WEBrick web server. A servlet is a chunk of code that handles requests directed to a given path on the web server. When the client browser sends a request to a URL that includes this path, the servlet is invoked. It can optionally extract information from the request, then provide data for the response to be sent back to the browser.

Class *BlogServlet* (Listing Four) is the servlet implementation of the blog. The < sign in the class definition specifies that the *BlogServlet* class is a subclass of WEBrick's *AbstractServlet*—you inherit all the basic servlet functionality. To handle incoming requests, the servlet implements a *do_GET()* method, which takes request and response objects as parameters.

The blog servlet doesn't need information from the request, and simply sets the response content to the 10 most recent files in the document tree. Notice here that I use *map()* again, this time to convert the array of *Article* objects to an array of the corresponding article bodies. I then use the method *join()* to convert this list into a single string with an *<hr>* element between each article.

You mount this servlet on the path /blog. Once you start the server running, you can access our blog using the URL http://localhost:2000/blog/.

Having written a basic blog, you can make its content available via an RSS feed.

RSS Summary

RSS is a simple XML-based messaging format. An RSS message contains a channel header followed by zero or more items; see Figure 1. The channel header identifies the source of the message, and includes a description, title, and the URL of the site serving the feed. Listing Five is the channel header for the BBC News RSS feed (which contains a summary of current news stories).

Following the channel header, an RSS message contains a set of items. Each item is intended to summarize an external piece of information—the RSS item contains a title, a brief description, and a link pointing to a fuller version of the information. To continue the BBC News example, you might find items each containing a synopsis of a news story and a link back to the BBC site where the full story is stored.

There are three major variants of RSS. Netscape created the original 0.90 specification. Later versions in the 0.9 series (0.91 through 0.94) were developed by UserLand Software. The RSS-DEV Working Group subsequently produced RSS 1.0, a total rethink that married RSS technology with RDF, a union that many felt was too complicated. As a reaction, UserLand produced RSS 2.0, which ignored the RDF flavor of 1.0 and instead built on the more successful 0.9 protocol. Unless you have a particular need for the data structuring capabilities of RDF, I suggest sticking with RSS 0.9x or 2.0.

Adding RSS Feeds

The RSS library supplied with Ruby 1.8.2 and later makes it easy both to parse and to create RSS information in 0.9, 1.0, and 2.0 formats. The library represents RSS data as a series of objects—for an RSS 0.92 feed, an RSS object contains a *Channel* object, and that channel object contains a collection of *Item* objects. The RSS-generating servlet (Listing Six) shows how these objects are used to create an RSS feed for this simple blog. You start by creating an RSS object—in this case, you ask for one that's compatible with the 0.9 specification. You then create a *Channel* object, populating it with the blog's title and a link to the top page of the web site. The *URI* class, also new in Ruby 1.8, makes this easy: You get the URI that was used to request the RSS feed and remove the path component, creating a link to the top-level page of the site. When this address is accessed, WEBrick automatically looks for an index.html file and displays its contents.

Each item in the RSS you generate corresponds to an article in the blog, so you can reuse the *Article.list()* method. The *each()* method iterates over this list, passing each article in turn to the associated block. Inside the block, you work out the URL for the article (by adding its path to the base URI of the site) and create and populate a new RSS *Item* object. This is then added to the list of items in the channel.

Finally, you fill in the response object, giving it an appropriate mime type and the RSS data as a body.

Other Types of RSS

There's no rule that says RSS has to contain blog entries, or that RSS has to be dynamically generated. In fact, one of my favorite uses involves creating a flat file containing an RSS summary of the last 10 commits to a CVS repository.

Each CVS repository contains a number of control files in the special CVSROOT project. One of these files, loginfo, can be

used to invoke an arbitrary program as files are stored back in the repository. Each noncomment line in the file contains a pattern and a command. If the pattern matches the directory being saved back into the repository, the corresponding command is executed. This command is passed information about the commit as parameters, and the log message supplied by users, along with a list of modified files, is passed to the command's standard input.

The Ruby program in Listing Seven is designed to be invoked by CVS as a loginfo script. It maintains a separate RSS file for each top-level repository in a site-wide directory. If this directory is accessible to a web server, the contents of this file can be disseminated over HTTP to RSS aggregators.

The CVS loginfo file that invokes this program contains a line similar to this:

```
ALL  /usr/local/bin/ruby /path/to/
                        commit2rss.rb %{}
```

This causes the Ruby script to be run ever ytime a commit is made to the repository. The *%{ }* parameter means that the path of the committed files are passed to the script. At the top of script, you extract the first component of this pathname. This is the top-level CVS module being committed, and you use this to generate the name of the RSS file in the drop directory.

The script keeps the 10 most recent commits in the RSS file. This means that information on the latest commit must potentially be merged with any existing data. Fortunately, Ruby's RSS library can parse RSS data, so we pass it the contents of any existing RSS file. Because this fails the first time the program is run, you open the file in an exception handler, creating a new RSS object if you can't populate one from existing data. Just to add some topical interest to the example, I've used Version 2.0 of RSS in this code.

After setting up the channel information, the code reads the CVS log message from standard input. Lines ending with a colon are headings, so I make them bold. I use the formatted log message to construct a new RSS item, which gets added to the channel. I then concatenate up to nine items from the original RSS file before writing the result back to disk. The net effect is that the RSS file now contains the new log message as the most recent item, followed by up to nine previous items. Figure 2 shows what this RSS feed looks like in the RSS aggregator, NetNewsWire. The panel on the left contains a list of RSS feeds that I read. At the top are several feeds that I generate from external sources using Ruby scripts. The CVS log summary for the Bookshelf repository is selected, and the list of recent commits is shown in the panel at the top-right. Below it is the log message for one recent commit. As authors commit changes to books, the panel updates, so I can see at a glance all activity in the repository.

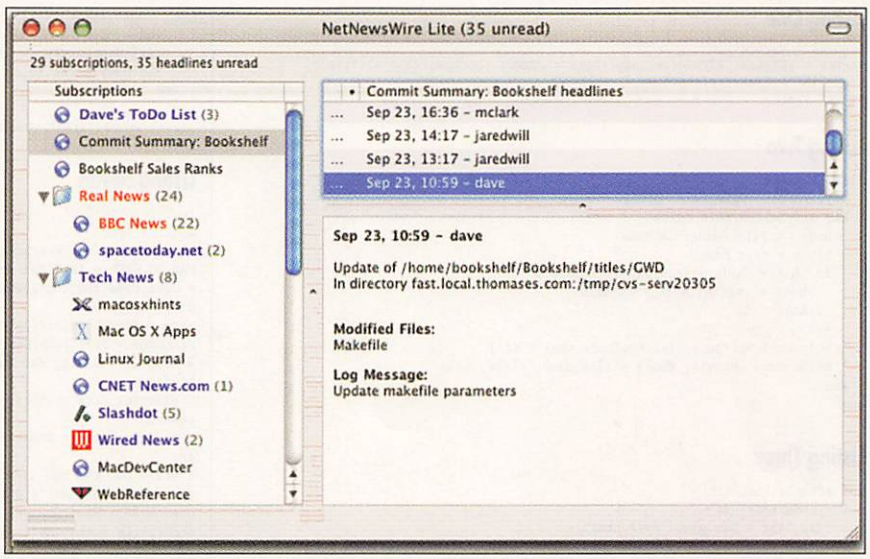


Figure 2: *An RSS feed in the RSS NetNewsWire aggregator.*

RSS to HTML

It is occasionally useful to be able to go the other way—taking information from an RSS feed and converting it into HTML. For example, a portal site might want to include a side box with a list of recent headlines from a news site. Because I don't want to fetch the RSS data every time the portal page is displayed, I generate the HTML for the side box periodically, and upload it into a known location on our web server. There, it can be included in the portal's page as a server-side include. The Ruby program in Listing Eight does this. Along the way, it also illustrates several other libraries now distributed as standard with Ruby.

The open-uri library illustrates the dynamic nature of Ruby's classes. When you include the library, it modifies the behavior of Ruby's standard *open()* method, letting it open URIs as if they were regular files. You use this capability to open an RSS feed (in this case, from my personal blog—http://pragprog.com/pragdave/). It reads the RSS into a string, then uses the RSS library to parse it.

Ruby 1.8 comes with a simple templating system that I use to convert each of the RSS items into an HTML fragment. This templating system takes a string containing a template and a hash containing key/value pairs. It looks for instances of *%name%* in the string, and replaces each with the value in the hash whose key is *name*. It can also generate repetitive entries—if the template contains lines between *START:name* and *END:name*, then the value corresponding to *name* in the hash is assumed to be an array, and the lines in the template are repeated for each entry in this array. I use this capability in the program to generate a list entry for each item in the RSS feed. The *map()* call takes the RSS item array and returns a new array, where each entry is a hash, and each hash contains the key's title, link, and description. I then use the template library to generate HTML from this, writing the result to a temporary file. (I could have taken a different approach to solve this problem, perhaps using XSLT to transform the RSS into HTML. However, doing this work in Ruby lets me showcase a few additional Ruby libraries.)

The last five lines in the listing use Ruby's Net::FTP library to connect to the portal web site and upload this HTML I just generated into the portal/side boxes directory. Notice how the library makes use of a Ruby block: The FTP connection is passed in to the block as a parameter, which uses it to transfer the file. When the code exits the block, the library is automatically closed.

Conclusion

It has been four years since I wrote the first article on Ruby for *DDJ*. In that time, the language itself has been relatively stable. The libraries, however, have matured tremendously. Out of the box, Ruby's libraries make it a mature player on today's Internet. Add to that third-party frameworks such as Ruby on Rails (http://www.rubyonrails.org/), and Ruby has become a serious contender as a language for knitting together the Internet.

DDJ

(Listings begin on page 30.)

Listing One

```
require 'webrick'
server = WEBrick::HTTPServer.new(:Port => 2000, :DocumentRoot => "html")
trap("INT") { server.shutdown }
server.start
```

Listing Two

```
class Article
  attr_reader :file_name, :title, :body
  def initialize(file_name)
    body  = File.read(file_name)
    title = file_name
    if body =~ %r{<title.*?>(.*?)</title>}m ||
       body =~ %r{<h1.*?>(.*?)</h1>}m
      title = $1
    end
    body.sub!(%r{<body.*?>(.*)</body.*}m) { $1 }
    @file_name, @title, @body = file_name, title, body
  end
end
```

Listing Three

```
def Article.list(dir)
  Dir.chdir(dir) do
    file_list = Dir.glob("**/*.html")
    sorted_list = file_list.sort_by {|name| File.stat(name).mtime }
    sorted_list.reverse[0, 10].map do |file_name|
      Article.new(file_name)
    end
  end
end
```

Listing Four

```
class BlogServlet < WEBrick::HTTPServlet::AbstractServlet
  HEAD = "<head><title>Simple Blog</title></head>"
  def do_GET(req, res)
    articles = Article.list(@server.config[:DocumentRoot])
    content  = articles.map{|a| a.body}.join("<hr />")
    res.body = "<html>#{HEAD}<body>#{content}</body></html>"
  end
end
server.mount("/blog", BlogServlet)
```

Listing Five

```
<rss version="0.91">
  <channel>
    <title>BBC News | News Front Page | World Edition</title>
    <link>
      http://news.bbc.co.uk/go/click/rss/0.91/public/-/2/hi/default.stm
    </link>
    <description>
      Updated every minute of every day - FOR PERSONAL USE ONLY
    </description>
    <language>en-gb</language>
    <lastBuildDate>Tue, 21 Sep 04 21:46:23 GMT</lastBuildDate>
    <copyright>
      Copyright: (C) British Broadcasting Corporation,
      http://news.bbc.co.uk/1/hi/help/3281849.stm
    </copyright>
    <docs>http://www.bbc.co.uk/syndication/</docs>
    <image>
      <title>BBC News</title>
      <url>
    http://news.bbc.co.uk/nol/shared/img/bbc_news_120x60.gif
      </url>
      <link>http://news.bbc.co.uk</link>
    </image>
    <!-- list of items ... -->
  </channel>
</rss>
```

Listing Six

```
class RssServlet < WEBrick::HTTPServlet::AbstractServlet
  def do_GET(req, res)
    my_uri = req.request_uri
    rss = RSS::Rss.new("0.9")
    chan = RSS::Rss::Channel.new
    chan.title = "My Blog"
    my_uri.path = ""          # link back to the top-level
    chan.link = my_uri.to_s
    rss.channel = chan
    Article.list(@server.config[:DocumentRoot]).each do |article|
      item = RSS::Rss::Channel::Item.new
      my_uri.path = "/" + article.file_name
      item.link  = my_uri.to_s
      item.title = article.title
      item.description = article.body
      chan.items << item
    end
    res['Content-Type'] = "text/xml"
    res.body = rss.to_s
  end
end
server.mount("/rss", RssServlet)
```

Listing Seven

```
require 'rss/2.0'
require 'etc'
DROP_DIR = "/var/www/pragprog/data/rss" # Where we create our RSS files
MAX_TO_KEEP = 10 # Max entries in the RSS file
# Just need the top-level project name
dir = ARGV.shift
repo = dir.split(%r(/))[0]
FILENAME = File.join(DROP_DIR, repo + ".rss")
# Read in existing rss (we'll write it out again to the new file)
begin
  existing_data = RSS::Parser.parse(File.read(FILENAME), false)
rescue
  existing_data = RSS::Rss.new("2.0")
end
# write out the new item and up to 9 old items
rss = RSS::Rss.new("2.0")
chan = RSS::Rss::Channel.new
chan.title = chan.description = "Commit Summary: #{repo}"
rss.channel = chan
# Read in the loginfo msg and highlight the headings
desc = ""
while line = STDIN.gets
  line.chomp!
  if line =~ /^[A-Z].*:\s*$/
    desc << "<p /><b>" << line << "</b>"
  else
    desc << line
  end
  desc << "<br />"
end
# new top item is current loginfo
item = RSS::Rss::Channel::Item.new
item.title =  Time.now.strftime("%b %d, %H:%M") + dir
item.pubDate = Time.now
item.description = desc
chan.items << item
# Then up to `n' items from old data
chan.items.concat existing_data.items[0, MAX_TO_KEEP-1]

File.open(FILENAME, "w") {|f| f.puts(rss.to_s) }
```

Listing Eight

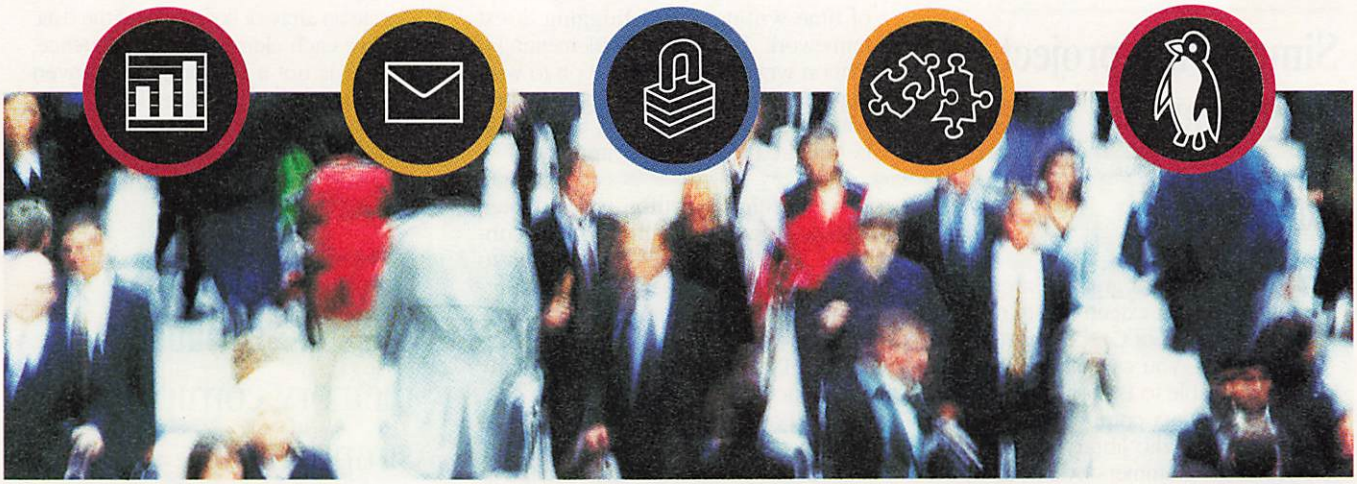
```
# Such down the top 'n' articles from an RSS feed and summarize for a web site
require 'open-uri'
require 'rss/0.9'
require 'rdoc/template'
require 'net/ftp'

TMP_FILE = "/tmp/topfive"
BLOG_URL = 'http://pragprog.com/pragdave/synopsis.rss?count=5'

TEMPLATE = %{
<ul>
START:entries
<li><a href="%link%">%title%</span></a>%description%</li>
END:entries
</ul>
}
open(BLOG_URL) do |http|
  result = RSS::Parser.parse(http.read, false)
  # Convert an array of RSS items into an array
  # of hashes
  entries = result.items.map do |item|
    {
      'title'       => item.title,
      'link'        => item.link,
      'description' => item.description
    }
  end
  File.open(TMP_FILE, "w") do |f|
    t = TemplatePage.new(TEMPLATE)
    t.write_html_on(f, 'entries' => entries)
  end
end
Net::FTP.open('www.pragprog.com') do |ftp|
  ftp.login('username', 'password')
  ftp.chdir('portal/sideboxes')
  ftp.put(TMP_FILE, 'topfive', 1024)
end
```

DDJ

LinuxWorld® CONFERENCE & EXPO

CONFERENCE: February 14 – 17, 2005 EXPO: February 15 – 17, 2005

HYNES CONVENTION CENTER • BOSTON, MA



LinuxWorld Conference & Expo is the world's leading and most comprehensive event focusing on Linux and Open Source solutions. This February, discover how companies across the globe have achieved higher profits and increased their productivity by utilizing Linux – the fastest-growing operating system in the world.

- **Hear** real-world experiences on how Linux and Open Source have been deployed and integrated throughout organizations.
- **See** and test drive the latest in Linux and Open Source technology and discover how using a Linux operating system will save your company time and money.
- **Learn** best practices and solutions by attending LinuxWorld's outstanding educational program featuring leading minds in the open source community.

WWW.LINUXWORLDEXPO.COM

Where OPEN MINDS Meet

Register Online With Priority Code: D0401

PLATINUM SPONSORS

ca Computer Associates

IBM

Novell

IDG WORLD EXPO

Extending Python

Simplifying project maintenance

GREG SMITH

One of Python's greatest strengths is that you can extend it with modules written in C or C++. By creating extensions, you can make existing C facilities available to Python programs; for example, the data compression facilities of the standard zlib library are available to Python programmers by means of the zlib module, which is simply a Python extension acting as a wrapper around the zlib API.

The process of creating such a module is well documented in the "Extending and Embedding" section of the Python manual (http://www.python.org/doc/current/ext/ext.html). It is also possible to implement new Python data types in C/C++ extensions. In this article, I only address C, but the concepts are equally applicable to C++.

There are three main reasons to write Python extension modules (as opposed to, say, solving the problem entirely in Python):

- You need the higher execution speed of C.
- You have an existing C module that you want to make available for Python applications.
- The C module may be your primary project, and you want to give it a Python API, allowing the testing and verification framework to be in Python. I've found myself in this situation fairly often. For example, when debugging and verifying a C module, I don't want to spend a lot of time writing and debugging a test framework. By quickly implementing a Python wrapper, I can move on to writing complex test frameworks in Python with much less overall effort than would be required for an "all-C" approach.

Greg is a senior DSP engineer at Silicon Optix. He uses Python for prototyping and verification of ASIC circuitry. He can be contacted at gsmith@siliconoptix.com.

When writing a Python-to-C adapter, most of the work usually involves converting Python data (the parameters to functions) into the basic data types required by the C function, then converting the return value, if any, to a Python object. The Python/C API provides many handy functions to help with these conversions. For instance, the *PyArg_ParseTuple* function (which is rather like *scanf*, but reading Python data types instead of strings) is useful for converting Python parameters into C data, while simultaneously checking the types.

However, there are many cases where the data conversion you'd really like requires a fair bit of code.

A Simple Example

Listing One (stats.c) is a simple C module that implements a function, which finds the mean and variance of an array of floats (the stats.h file contains just a function prototype and is not shown). Listing Two is statsmodule.c, a Python extension module based on stats.c.

Since the *find_variance* function accepts a variable-sized input array, its Python parameter list cannot be directly handled by *PyArg_ParseTuple*. The interface function has been written to accept a list of floats; in fact, to be more "Pythonic," it accepts any sequence of objects that can be converted to floats.

Refer to the function *stats_find_variance* in Listing Two. To perform the conversion, the C code must:

- Verify that there is only one parameter, and that it is a sequence.
- Obtain the length of the sequence and verify that it is nonzero (since the C function to be called requires that).
- Allocate an array of floats to hold the data.
- Retrieve each element of the sequence, and if it is not a Python *float*, convert it to a *float* while checking for an error in the conversion; then, store the result in the array.

> "It should not be possible to cause memory corruption simply by writing incorrect Python code"

If all of these operations succeed, the code calls the *find_variance* function in the stats.c module, and then converts the results (two doubles) back into Python floats. In addition, the code must carefully maintain the reference counts on Python objects to avoid memory leaks or similar problems.

As you can see, the code to perform this conversion amounts to more than half of the program. In fact, I wrote most of the rest of this module by simply pasting the examples from the Python documentation, then changing "spam" into "stats." In addition to the *find_variance* function, the module contains a table of pointers to functions, which is used by the Python internals to find the entry points; and an initialization routine, initstats, makes this table available to Python.

Usually, I finish writing the C/Python interface quickly, and move on to something else. This means avoiding the kind of code that can be seen in Listing Two, which is fairly involved. It needs to check

Love Your Analyst

www.guibot.com

- Deliver painless GUI prototyping
- Use Extended Activity Semantics to bridge the gap

XAS compliments prototyping with a detailed, accurate and complete specification of user and interface behavior. With Guibot's XAS enhanced tools, prototype interfaces that fully realize business requirements can be automatically generated and documented, making collaborative development a breeze.

- Quantify end-user scope
- Accelerate acceptance
- Keep everyone on the same page

Discover a better way to specify user-requirements and to prototype user-interfaces

- Guibot Rose Rational Rose plug-in
- Guibot Xtreme stand-alone
- Guibot Services we make it even easier

Better software starts with Guibot

- Download evaluation versions at www.guibot.com

Guibot is a trademark of Nuvotec, Inc. Rational Rose is a registered trademark of International Business Machines Corporation

(continued from page 32)
for several error conditions, even for a simple case such as this.

Using a Python Wrapper Layer

The solution is to do the work in Python. Since the underlying C function needs a pointer to an array of floats, why not require the Python code to supply a contiguous chunk of memory containing that array? The data could be contained in a Python string, for instance. There are several ways to do this, but it puts you in an odd position if, to obtain statistics on a set of floating-point numbers, they must be first converted to a binary string format.

To fix that problem, add a layer of Python to do the work. Listing Three is statsimodule.c, a new extension module that is the same as statsmodule.c, except that the *find_variance* function accepts a string instead of a sequence of floats. The function is simpler—it can now use *PyArg_ParseTuple* to obtain the address and length of the supplied string data (while checking that the parameter is of suitable type). It then performs a simple length check and calls the *find_variance* function, passing it a pointer to the data supplied by the caller.

The Python wrapper code, stats2.py (Listing Four), contains a function *find_variance* that does the conversion and passes the result to the built-in function in the statsi module (a Python function implemented in C is referred to as "built-in," even when it is in an extension module). To convert the parameter to a string, it creates an instance of the Python type *array.array* with typecode *'f'*; that operation packs the data internally into machine floats, exactly as desired. It is then possible to convert the array to a string. Instead, I pass the array directly to the built-in *find_variance* function. This works because *array.array* implements the *buffer* interface—it is based on a memory buffer, and can identify the location and length of that buffer via the standard buffer protocol.

So, both of these code snippets give the same result:

```
>>> import stats
>>> print stats.find_variance( [ 1.0, 1.5, 1.8, 2.12 ] )
(0.17007496588230353, 1.6049999594688416)

>>> import stats2
>>> print stats2.find_variance( [ 1.0, 1.5, 1.8, 2.12 ] )
(0.17007496588230353, 1.6049999594688416)
```

However, the stats module is all C, including a fairly complex conversion routine, while the stats2 module is split into two layers. Also, I've removed the *docstring* from *find_variance* in statsimodule, instead placing it in stats2.py where it is visible at the upper layer.

Splitting code up like this is a popular approach; for instance, the socket module in the standard Python library is written in Python and uses a lower level C module, *_socket*. However, the socket module is not just a wrapper. It provides additional higher level functions.

I find that splitting the code up like this can create a maintenance problem; there is now an interface between stats2.py and statsimodule.c, and it is rather an odd interface. In fact, this is the whole point of the exercise: to use an unusual (some might just say crude) interface to make the C programming easier, but to supply a conversion layer so that Python programmers don't need to use that unusual interface. If I have to maintain that interface (which could mean documenting it somewhere other than in the comments of the source, or taking steps to make sure that both files are always updated together) then I've made extra work, which is what I've been trying to avoid in the first place. Code used as part of a test framework is frequently modified and run during development. Changes to the Python code generally take effect immediately, while changes to the C module do not take effect until it is rebuilt. So the potential risk of problems arising from incompatible versions is higher than usual.

Combining the Wrapper Layer with The Extension Module

Version incompatibility problems can be avoided by putting the C code and the Python code in the same file. Fans of the Obfuscated C contest are, no doubt, now scribbling down *#if 0* and figuring out how to make the same file acceptable to both C and Python. This is pretty easy, actually, but it's not what I meant.

Built-in Python modules (including those based on C extensions) are of exactly the same basic Python type as modules created from pure Python code. Both types have a dictionary attached, which contains the global namespace of the module; the difference lies only in the kinds of function objects that are placed in this namespace when the module is constructed. So rather than putting the wrapper function in a separate Python file, you can include it in the C module.

The program stats3module.c (available electronically; see "Resource Center," page 5) is identical to statsimodule.c, except that the entry point is called *_find_variance* instead of *find_variance*. This lets you introduce *find_variance* later. Also, there are a few lines of Python code in a *char[]* variable called *py_code_string* and some extra code in the module initialization function.

The extra code simply obtains the namespace dictionary for the module and calls *PyRun_String* to execute the Python code. The module dictionary is supplied as the context for the execution, which means that all *global* values defined in the Python code wind up in the module's namespace. It also means that any Python functions defined by this operation will have, as their global namespace, the module's namespace.

So, referring to the contents of *py_code_string*, it looks a lot like stats2.py, except that it doesn't need to import a lower level module, and it refers to the lower level function directly as *_find_variance*, rather than as *statsi.find_variance*.

There is another step performed during the module initialization: Before calling *PyRun_String*, I obtain a reference to the *__builtins__* module and place this reference into the *stats3* module namespace. This module is part of the namespace of any pure-Python module; without it, all the built-in objects and functions (*None*, *abs*, *len*, *range*, *open*, to name a few) are unavailable. In fact, without this entry, you can't even execute an *import* statement because that is hooked through the *__import__* function in *__builtins__*.

Another little detail: The Python code in *py_code_string* imports array as *_array*, to keep it from being included when "from stats3 import *" is done. This statement includes all names not starting with an underscore; in the case of *stats3*, that means just *find_variance*.

So, *stats3* looks just like *stats* and *stats2*, from the Python side; like *stats*, it's implemented as a standalone C extension, but, like *stats2*, it includes a Python wrapper layer.

When initially creating and debugging such a module, I sometimes replace the entire *py_code_string* contents temporarily with the single statement

```
execfile('mycode.py')
```

This lets me compile the extension just once, and debug the contained Python code by changing the contents of the file mycode.py. When it's working, I can just paste it back into the C string.

A Note on Type Conversions

There is an interesting difference between stats (implemented in statsmodule.c) and the other two implementations:

```
>>> import stats
>>> stats.find_variance( [ 10.0, 11, "12"] )
(0.6666666666666663, 11.0)
```

If you try this with the other modules, it raises an exception. statsmodule.c simply looks at each of the elements in the supplied sequence, and if the element is not a *float*, it tries to convert it to one. The string "12" can be converted to a *float*, 12.0. This is rather more implicit type conversion than

most Python programmers are used to. Here is a more dangerous example:

```
>>> stats.find_variance( ("1234",) )
(0.0, 1234.0)
>>> stats.find_variance( ("1234") )
(1.25, 2.5)
```

The first example shows a tuple of strings—just one string—being passed to the function, so the mean is 1234.0 and the variance is zero. The second example shows a string being passed to the function—the parentheses without the comma have no effect. The string is, according to Python type conventions, a sequence of one-character strings, each being a single digit. So the second example finds the mean and variance of the four values 1.0, 2.0, 3.0, 4.0. This is a hazard of having too many automatic type conversions—a simple programming error may result in behavior different from what was expected, where you would rather get an exception.

If you actually want the other modules to have this behavior, it's easy to add, since you can do it in the Python layer. Just change this:

```
if type(fltlist) is not list:
   fltlist = list(fltlist)
```

to:

```
fltlist= map(float, fltlist)
```

This breaks the supplied sequence into its components, explicitly converts each one to a Python *float,* and builds a list of the results. Making a similar kind of change to C code could require substantially more work.

Calling Python Code from C

stats4module.c (available electronically) implements the *find_variance* function directly in C again; but this time it takes its parameter and passes it to a pure Python function attached to the module, and lets that do all the work of the conversion. That function, *_flts_to_array*, converts whatever is passed to *find_array* into an *array.array* of C floats, and returns that. It enforces the minimum length of one element and returns the result inside a tuple, so that the C code can pass it directly to *PyArg_ParseTuple* to get its address and length. The *args* parameter, passed to the C function *stats4_find_variance*, is a reference to a Python tuple containing all of the parameters passed to the function. The same protocol is used when calling an arbitrary Python object from C, using *PyObject_CallObject*; the first parameter is the object to call, while the second is a tuple of the parameters. If someone improperly calls *find_variance* with zero parameters, or more than one, then an error will occur when *PyObject_CallObject* attempts to call *_flts_to_array* with the same parameter set. It is possible to catch this error in the Python code, and to provide more detailed error descriptions for other conditions, if desired.

> "Rather than putting the wrapper function in a separate Python file, you can include it in the C module"

If your C module uses callback functions, your wrapper needs to implement these C functions; the callback function generally needs to manipulate Python data. Calling Python code from these C functions is often the easiest way to handle this.

Reporting Errors

It is fairly easy to check the parameters in such a way that an exception is raised when there is a problem—many of the possible error conditions cause the normal conversions to throw exceptions.

It is generally more work to make sure that the exception messages are detailed and directly meaningful to API programmers. In the previous example, if the parameter count to *find_variance* is incorrect, the exception indicates that the number of parameters passed to the internal function *_flts_to_array* is incorrect, which may certainly be confusing. This issue is fairly common in Python; it is convenient to allow lower levels of code to check for problems and raise exceptions. The drawback is that you end up with possibly confusing error messages; you need to look at each line on the stack traceback to find where the problem really originated. With extra work, you can either check these conditions at a level closer to the API, or catch exceptions thrown by lower level code and restate them in better terms.

As I mentioned, I often write Python-C interfaces in order to create test frameworks in Python. I don't need to be too concerned about API users being confused by error messages, because this code does not wander very far from my desk. It is, however, important that the type and range checking is thorough. If, during testing, the application throws a segmentation fault, I want to be sure that this did not occur simply because the test framework actually passed bad data through the Python API. In other words, it should not be possible to cause memory corruption simply by writing incorrect Python code. This principle should be applied to all C/Python interfaces; much of the convenience of using Python is lost if you need to use C-language debugging techniques to find problems in your Python code.

User-Defined Python Types

If your C extension implements a Python data type, the interface between the Python wrapper and the C code can become more complex, and these techniques can be even more useful. For instance, the Python wrapper can define a Python class, which is based on the type defined in the C module, and supply implementations for methods that are more easily coded in Python. The methods of the base class, being written in C, could use the Python/C API calls to ensure that when member functions written in C are overridden in a derived class, the correct function is called even from C code. It is useful to be able to split such an implementation into a C layer and a Python layer, but it may be cumbersome to have these in separate source files.

The Downside

There are only two real drawbacks to this approach: The Python source code must appear in quoted strings in the C file, making it inconvenient to maintain, and the Python layer is "compiled" to Python bytecode every time the module is loaded. In a conventional approach, with a separate .py file, Python compiles the code once, and save the results in a .pyc file for later use. So, the technique is mainly applicable when the Python code is quite short. This is not unusual because significant data conversions can be performed in a few lines of Python.

DDJ

Listing One

```
/* ---------- stats.c --------------
 * simple example of a C module
 */
#include <math.h>

#include "stats.h"
/* given a set of pointers to 'n' floats (>=1), find the mean and variance.
 * The variance is returned directly; the mean via a pointer.
 */
double
```

(continued on page 36)

(continued from page 35)

```
find_variance( float const *data, int npts, double *mean_p)
{
    int i;
   double sum = 0;
    double sumsq = 0;
    double mean;
    /*
     * there are more numerically stable ways to do this, but
     * that's a different article...
     */
    for( i = 0; i < npts; ++ i ){
        double x= data[i];
        sum += x;
        sumsq += x*x;
    }
    mean = sum/npts;
    *mean_p = mean;
    return ( sumsq - mean*sum )/npts ;
}
```

Listing Two

```
/* ---------- statsmodule.c -------------- */
/* python extension based on stats.c       */

#include <Python.h>
#include "stats.h"

static char stats_find_variance_doc[] =
"Given a sequence of floats, find the standard deviation and mean";

static PyObject *
stats_find_variance(PyObject *self, PyObject *args)
{
    double mean, var;
    PyObject * flt_list;
    PyObject * lst_element;
    int n,i;
    float *tmp_arr;
    /* must be a single parameter which is a sequence of floats. */
    if( PySequence_Size(args)!=1 ){
        flt_list = 0;
    }else{
        flt_list = PySequence_GetItem(args,0);
    }
    /* at this point either flt_list is NULL, or
     * it's the param and we own a reference to it
     */
    if( flt_list ==0  || !PySequence_Check(flt_list)){
        PyErr_SetString( PyExc_TypeError,
                "find_variance accepts a list of floats");
        Py_XDECREF(flt_list);
        return 0;
   }
    /* we know it's a sequence. Get its length, and allocate an
     * array of floats. Length of 0 is an error.
     */
    n = PySequence_Size( flt_list );
    if( n < 1 ){
        PyErr_SetString( PyExc_ValueError,"find_variance: empty list");
        Py_XDECREF(flt_list);
        return 0;
    }
    tmp_arr= PyMem_Malloc( n * sizeof(float));

    if( !tmp_arr ){
        PyErr_SetString( PyExc_MemoryError, "out of memory");
        Py_DECREF(flt_list);
        return 0;
    }
    /* now fill in all the floats. If any operation fails, we have to free
     * everything. The code is structured so that if something returns NULL,
     * it will drop down to the single error test. We don't need to
     * set the exception, since the failed operation will do it.
     */
    for( i = 0; i < n ; i++ ){
        lst_element = PySequence_GetItem( flt_list, i );

        if( lst_element != 0 && !PyFloat_Check(lst_element)){
            /* succeeded, but not a float - convert it */
            PyObject *new_obj= PyNumber_Float( lst_element );
            Py_DECREF(lst_element);
            lst_element = new_obj;  /* may be 0 */
        }
        if( lst_element != 0 ){
            tmp_arr[i] = (float)PyFloat_AsDouble(lst_element);
            Py_DECREF(lst_element);
        }else{
            Py_DECREF(flt_list);
            PyMem_Free(tmp_arr);
            return 0;
        }
    }
    Py_DECREF(flt_list);    /* don't need that any more */
    /* call the actual C routine with all the floats */
    var = find_variance( tmp_arr, n , & mean );
    /* free our tmp array and return results */
    PyMem_Free(tmp_arr);
    return Py_BuildValue("dd", var, mean);
}
/* table of module methods. If the module has multiple
 * Python-callable functions, they are all listed here.
 */
static PyMethodDef StatsMethods[] = {
{"find_variance",  stats_find_variance,
    METH_VARARGS,  stats_find_variance_doc},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
/* The module initialization function */
PyMODINIT_FUNC
initstats(void)
{
    Py_InitModule("stats", StatsMethods);
}
```

Listing Three

```
/* ---------- statsimodule.c ---------------- */

#include <Python.h>
#include "stats.h"
/*
 * This is an extension implementing the 'stats'
 * module which is intended to be wrapped by a
 * python module; it provides a simple internal
 * interface.
 *
 * the 'find_variance' entry point accepts
 * a single parameter, which is expected to
 * be a read-only buffer of floats (at least one)
 */
static PyObject *
statsi_find_variance(PyObject *self, PyObject *args)
{
     double mean, var;
     char *data_ptr;
     int nbytes,n;

     /*
      * expect a single parameter, which is a string
      * or buffer, containing value in native order.
      * The API call below gets us the address and len
      * of the string, or fails.
      */
     if (!PyArg_ParseTuple(args, "s#", &data_ptr, &nbytes))
        return NULL;
     /*
      * check the len - it must be a multiple of sizeof(float)
      */
     n = nbytes/sizeof(float);
     if( n <= 0 || n*sizeof(float) != nbytes ){
          PyErr_SetString( PyExc_ValueError,
                    "find_variance: bad length");
          return 0;
     }

     /*
      * call the actual C routine with all the floats.
      * Note: there is a risk here that the data will not
      * be aligned on a suitable boundary for floats. This
      * is not an issue on X86 hardware; also, the Python
      * wrapper can guarantee this, if needed, by using
      * array.array type instead of a string.
      */
     var = find_variance( (float const*) data_ptr, n , & mean );
     /*
      * return results
      */
    return Py_BuildValue("dd", var, mean);
}
/*
 * table of module methods. If the module has multiple
 * Python-callable functions, they are all listed here.
 */
static PyMethodDef StatsiMethods[] = {
{"find_variance",  statsi_find_variance,
    METH_VARARGS,  ""},
    {NULL, NULL, 0, NULL}        /* Sentinel */
};
/*
 * The module initialization function
 */
PyMODINIT_FUNC
initstatsi(void)
{
    Py_InitModule("statsi", StatsiMethods);
}
```

Listing Four

```
#---- stats2.py ----
# same as 'stats', but done as a wrapper around a C
# extension (statsi)
#
import statsi
import array

def find_variance( fltlist ):
    "Given a sequence of floats, find the "\
        "standard deviation and mean"
    try:
        if type(fltlist) is not list:
            fltlist = list(fltlist)
        data = array.array( 'f', fltlist)
    except (TypeError,ValueError):
        raise ValueError, 'bad parameter to find_variance'
    return statsi.find_variance(data)

#
# don't export array and statsi
#
__all__ = ['find_variance']
```

DDJ

Mixins Without Multiple Inheritance

Extending REALbasic—and other languages, too!

JOE STROUT

A mixin class is a small class designed to be blended with other classes through multiple inheritance. Usually, mixin classes are not intended to be instantiated directly, but instead just add a few extra functions or properties to some other class.

I recently needed a mixin class in an editor application. However, this application was being written in REALbasic, which, like Java, supports only single inheritance. REALbasic, from Real Software (the company I work for), is a Basic-based, cross-platform development environment that runs on and targets Windows, Linux, and Macintosh. (For more information, see http://ww.realsoftware.com/.) However, REALbasic also provides some language features that let you provide mixin functionality and syntax in a different way. The technique I present here can be adapted to other languages as well, though in a somewhat more limited fashion.

In my application, objects of a number of different classes needed to be configurable via a Property Editor window. The set of classes that had editable properties did not fit well onto the existing class hierarchy, which was based on more fundamental functionality. In other words, the "has properties" concern cut across the concerns of the inheritance tree. That, and the fact that the property editor needed only a handful of methods, made this an attractive candidate for a mixin.

Joe is a senior software engineer at REAL Software. He can be contacted at joe@strout.net.

In languages with multiple inheritance (C++ or Python, for instance), I might have simply defined a small abstract *PropertyHolder* class, with some pure virtual methods and some additional methods defined in terms of the first set. For example, *PropCount*, *PropName*, and *PropValue* must be implemented by each *PropertyHolder*, but *PropIndex* would be implemented by the mixin itself; see Table 1. I could then simply add this as an additional base class wherever needed (watching out for any problems caused by inheriting this base class twice). But since REALbasic does not allow multiple inheritance, this approach was not an option.

One alternative to a class is an interface. An interface is like a class that contains only pure virtual methods; any class that implements an interface must implement all those methods. REALbasic, as well as languages such as Java, lets a class implement any number of interfaces. Because an interface can't itself contain any implementation, the problems of inheriting the same implementation twice are neatly avoided.

An interface isn't quite the same as a mixin class, though. Mixins do more than just define an interface; they provide some functionality, too. In my case, the *PropIndex* method and the *PropValue* methods that work by name, would have to be duplicated by every class that uses the interface, with all the maintenance problems that would cause.

REALbasic has an unusual language feature that provides a solution to this problem—the extension method. Extension methods are like global methods, except that they are inserted into the namespace of another type. That type may be atomic (*Integer*, for instance), a class type, or in this case, a class interface. Combining extension methods with a class interface results in the functional and syntactic equivalent of a mixin class.

The Solution

I started by defining an interface called *PropertyHolder* (Figure 1), containing just the low-level methods (ones that would be pure virtual in a C++ mixin class); see Listing One (the complete sample code in REALbasic binary project format is available electronically; see "Resource Center," page 5). The next step is to make a module to contain the extension methods. In REALbasic, method definitions and properties that aren't part of a class live in a module, which is something like a namespace or source file in other languages.

> "The basic principles used here could be applied to other languages, such as Java"

Methods in a module may be global to the whole project, or they may be protected or private members of the module's namespace. Or, by using the *Extends* keyword, they may insert themselves into another namespace.

I created a module called "PropHolderExtensions." When you intend a module to contain only extension (or global) methods, then its name isn't very important, but naming it this way made its purpose more clear when browsing the project. The extension methods could also be put into some other module; for example, I've occasionally made a module called "Globals" to hold all my global methods, and the extension methods would work fine there. But again, that would make their purpose more obscure, and make it harder to transfer the mixin to another project.

Within my *PropHolderExtensions* module (Listing Two), I added the methods

that extend the class interface. A class extension method is like any other global method, except that its first parameter

```
(a)
resultCount = obj.PropCount
resultName = obj.PropName( index )
resultValue = obj.PropValue( index )
obj.PropValue( index ) = newValue

(b)
resultIndex = obj.PropIndex( name )
resultValue = obj.PropValue( name )
obj.PropValue( name ) = newValue
```

Table 1: *Methods of the* PropertyHolder *mixin. (a) Abstract methods are declared by the mixin interface but must be defined by implementors of that interface. (b) Concrete methods are provided by the mixin itself.*

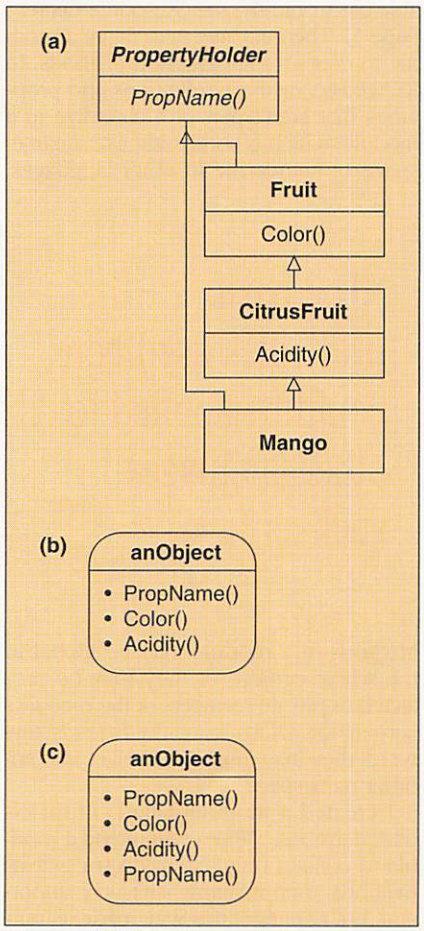


Figure 1: *(a) In the partial class hierarchy shown, class Mango implements the* PropertyHolder *interface twice— once directly and once via its ancestor Fruit. (b) In REALbasic, a Mango object contains only one entry in its* vtable *for the* PropName *function. (c) In C++, a Mango object contains two* PropName *entries in its* vtable*, and which implementation is invoked depends on the static type of the object reference.*

uses the *Extends* keyword, and identifies the object on which the method will act, like so:

```
Function PropIndex( Extends ph as
  PropHolder, propName as String ) as Integer
```

Now, wherever I need to make one of my existing classes a property holder, I simply add the *PropertyHolder* interface, and implement the methods it defines. (REALbasic supplies the method declarations for me, and if I forget to implement one, it generates a compile-time error.) I can then use not only the interface functions just implemented, but also the higher level functions, all with the same syntax. For example, if *w* is a *WindowClass* object that implements *PropertyHolder*, then I could write:

```
Dim idx as Integer
idx = w.PropIndex( "Title" )  // (1)
w.PropValue( idx ) = "Demo"  // (2)
```

Here, the assignment in line (2) is something implemented directly in *WindowClass*, but the index look-up in line (1) is not; it's part of the mixin.

Semantics

This combination of an interface and a set of extension methods has well-defined semantics. First, consider the methods defined by the interface. These are virtual, but an interface method always occupies just a single *vtable* slot, even if an interface is declared at multiple points in a class's ancestry; see Figure 1(b). So no matter what the static type of an object reference, a call to one of these methods will always invoke the most specific version, preserving polymorphism. This is somewhat different from the equivalent C++ case, where the particular method invoked may depend on the static type of the object reference; see Figure 1(c).

Next, consider methods added to the interface via the extension mechanism. These are really just global methods with a bit of syntactic sugar, and as such, they are nonvirtual. But that means that, again, it doesn't matter what the static type of a reference might be; the same extension method will be invoked regardless. Further, any methods invoked by those methods on the object will be properly polymorphic as previously noted. In a C++ multiple-inheritance mixin, the static type determines which copy of the mixin method is invoked, and that in turn affects which of the low-level functions are called by the mixin code.

For these reasons, the technique described here isn't just a workaround for the lack of multiple inheritance—it's actually better in some ways in that it neatly avoids a class of errors that can occur with the standard mixin approach.

Adapting to Other Languages

The basic principles used here could be applied to other languages, such as Java. You would define an interface (or class containing only pure virtual methods) and an associated set of global methods. Most of the benefits described here would apply.

However, without the *extends* keyword, the global methods would have to be accessed with a syntax different from the interface methods. That will make it more difficult to change your mind about which methods should be implemented by users of the mixin, and which can be provided by the mixin itself, since such a change will require updating all callers of the code.

Specializing a Mixin

One drawback to this technique is that a class can't easily override the mixin code. For example, suppose I have a property holder class that has a large number of properties and can look one up by name very quickly (perhaps through a hash table). The generic *PropIndex* method provided by the extension may be too slow in this case, but because it is an extension method, you can't override it as you could if using a traditional mixin.

There is a solution in such cases, but it requires a few extra steps. For this example, I would start by defining an interface called *FastPropHolder*, with a *PropIndex* method that accepts a property name and returns an integer. My class with the hash table would implement this interface as well as *PropertyHolder*. Then, the *PropIndex* extension method would be modified to check for objects that satisfy the *FastPropHolder* interface, and make use of it when available (see Listing Three).

Defining another interface and changing the mixin code is certainly not an ideal way to specialize a mixin. But in practice, I have rarely found this to be a problem—usually it is clear which methods should be implemented by the mixin itself, and which should be implemented by mixin users. In the occasional case where a specialization is needed, the solution here is quite serviceable.

State Data

In most cases, mixins don't need to keep any per-object state. In my example, there is nothing in the mixin methods defined by *PropHolderExtensions* that require any data from the *PropHolder* objects. But occasionally, this need does occur; for example, in a game project, a mixin might be used to make some classes of objects react to gravity, and the mixin code that

causes an object to fall might need some extra data about the object, such as its terminal velocity.

In this case, the easiest solution by far is to make a getter (and/or setter) for that data part of the mixin interface. Then, any class that uses the mixin will be responsible for storing or calculating the required data in some way, and the mixin code will work by simply getting it via the interface. The only drawback to this approach is that the state data accessors become a public part of those classes, usable not only by the mixin but by other code as well.

There is another approach to the mixin state that is worth mentioning. The module containing the mixin code could have a private hash table (in REALbasic, a *Dictionary* object) that maps mixin objects to the corresponding state data. But this raises issues of notifying the module when an object is created or destroyed, so that the associated state data can be updated accordingly. I mention this approach for the sake of completeness; in most cases, deferring storage of the state data to the implementor classes is a better solution.

Conclusion

It is commonly thought that mixin classes are a technique particular to languages with multiple inheritance. You now know how to make the functional equivalent of a mixin in single-inheritance languages, and gain some clearer semantics in the process. In REALbasic, in particular, this can be done in a way that preserves the syntactic convenience of standard mixins, and provides flexibility in deciding the mixin interface.

DDJ

Listing One

```
Interface PropertyHolder
  Function PropCount() As Integer
  Function PropName(index As Integer) As String
  Function PropValue(index As Integer) As Variant
  Sub PropValue(index As Integer, assigns newValue As Variant)
End Interface
```

Listing Two

```
Function PropIndex(extends obj As PropertyHolder, name As String) As Integer
    Dim i As Integer
    for i = 0 to obj.PropCount - 1
       if obj.PropName(i) = name then return i
    next
    return -1
End Function

Function PropValue(extends obj As PropertyHolder, name As String) As Variant
    Dim idx As Integer = obj.PropIndex( name )
    if idx > 0 then
       return obj.PropValue( idx )
    else
       return nil
    end if
End Function

Sub PropValue(extends obj As PropertyHolder, name As
                                    String, assigns newValue As Variant)
    Dim idx As Integer = obj.PropIndex( name )
    if idx > 0 then
       obj.PropValue( idx ) = newValue
    else
       raise new KeyNotFoundException
    end if
End Sub
```

Listing Three

```
Function PropIndex(extends obj As PropertyHolder, name As String) As Integer
    if obj IsA FastPropHolder then
       return FastPropHolder(obj).PropIndex( name )
    end if

    Dim i As Integer
    for i = 0 to obj.PropCount - 1
       if obj.PropName(i) = name then return i
    next

    return -1
End Function
```

DDJ

Just Released from C/C++ Users Journal: CUJ CD - 7
C/C++ Users Journal
Advanced Solutions for Professional Developers
© 2004, C/C++ USERS JOURNAL, CMP Media LLC, All rights reserved.
CMP
CD 7
January 1990 through December 2003
This fully searchable CD-ROM delivers over 165 issues of content from C/C++ Users Journal from January 1990 through, and including the December 2003 issue of C/C++ Users Journal, plus all issues of the Java Solution Supplement and all columns of CUJ's online C++ Experts Forum. Save time, just cut and paste information right into your work in progress, 24/7, without worrying about long download times.
C/C++ Users Journal
Advanced Solutions for Professional Developers
CMP
United Business Media
TO ORDER:
Online:
www.ddj.com/cdrom/
US Toll-Free:
1-800-444-4881
International:
1-785-838-7500
Fax:
1-785-838-7566

printf Revisited

Picking up where *IOStreams* leaves off

WALTER BRIGHT

I love C's *printf*. It's one of the original reasons why I switched from Pascal to C 20-plus years ago. It's quick and easy to bang out. It never complains, it's always faithful, it just does its job. It's just so darned useful and convenient.

But I have to admit it, *printf* has well-known faults:

- It's not typesafe. There's no checking to see if the format specifications and the argument types line up, or even if the arguments exist. Getting this wrong results in bizarre output and/or crashes.
- It doesn't work well at all for generic programming. For example, you must know the underlying type of a *typedef*ed argument to put in the right format for it, which defeats much of the purpose of using a *typedef*. This is especially a problem with templated code.
- The format string mixes together both the type of the argument(s) and how to print them.
- Poor character set handling. *printf* and *wprintf* cannot be interleaved on the same stream. This tends to wind up forcing you to choose between *char* and *wchar* throughout the app. Various wretched approaches like *<tchar.h>* attempt to mitigate this.
- Nonportable universal character (UTF) support. *wprintf* tries to deal with this, but fails because *wchar_t* is implementation defined.
- *printf*'s cousins (like *sprintf*) suffer from buffer overflows unless you are very, very careful. *snprintf* suffers from the guess-what-size-is-needed-in-advance kludge.
- It's not able to deal with user-defined types.
- There are too many confusing format characters, made worse in C99. There are so many that I have to refer to the manual on them, even though I've implemented a C99 *printf* for the Digital Mars C runtime library.

Walter is the author of the D language and the Digital Mars C/C++ compiler (among others). He can be reached at http://www.walterbright.com.

The most noticeable attempt to fix these faults was C++'s *IOStreams*, which itself has undergone two major overhauls, and credibly solves some of *printf*'s problems. But in the process, we lost what was so cool about *printf*. What was short and sweet turned long and clumsy. For instance, "Hello world" looks fine:

```
printf("Hello world!\n");
std::cout << "Hello world!\n";
```

but when there are more arguments, it starts looking dodgy:

```
printf("Elapsed time for %s is %d %s\n", foo,
etime, units);
std::cout << "Elapsed time for " << foo << " is "
<< etime << " " << units << std::endl;
```

But *IOStreams* really gets beaten with the ugly stick when you try to use something other than the default formatting:

```
printf("%d|%-8d|%d\n", 123, 456, 789);
std::ios_base::fmtflags flags_save = std::cout.flags();
std::cout << 123 << '|' << std::left
    << std::setw(8) << 456
    << "|" << 789 << std::endl;
std::cout.flags(flags_save);
```

Even worse, formatting with *IOStreams* introduces problems with exception safety and thread safety with respect to stream state, since the *std::left* relies on a sticky global state.

And finally, *IOStreams* inherits all of C's problems with UTF characters.

The D programming language (http://www.digitalmars.com/d/index.html) attempts to retain the great things about C and C++ while fixing the faults. Therefore, *printf* needs to be fixed to resolve the acknowledged problems while retaining its handy utility. Few other issues with D generated more debate, heat, ideas, and volume on the D newsgroup (news.digitalmars.com).

The obvious root of *printf*'s problems is the lack of type information corresponding to the *variadic* function arguments. In D, *variadic* functions are passed a hidden argument that is an array of the types of all the *variadic* arguments. This gives a free hand in designing a replacement—*writef*— and having these functions corresponding roughly to *printf*, *fprintf*, and *sprintf*:

> "Formatting with *IOStreams* introduces problems with exception safety and thread safety"

```
void writef(...);       // write arguments to
                        // stdout
void fwritef(fp, ...); // write arguments to fp
char[] format(...);    // create a string and
                       // write the arguments to it
```

What happened to the format string as the first argument? It isn't necessarily needed. The argument scanner in *writef*'s implementation scans each argument in order. If it is a *char[]*, it is interpreted as a format string; otherwise, it is printed in the default manner appropriate to its type:

```
writef(); // prints no characters
writef("hello", " world\n");    // prints 'hello
                                // world\n'
writef(123, 8.60, '@');         // prints '1238.6@'
```

If an argument is an instance of a class derived from *Object*, the class's *toString()*

method is called to get the corresponding string to print:

```
class Man
{
  char[] toString() { return "I am a Man, not an ape";
}
}
void foo(A a)
{
  writef(a);  // prints 'I am a Man, not an ape'
}
```

writef's format strings are greatly simplified because they no longer have to do double duty as type specifications. Any string not under the control of a previous format is interpreted as being a format string. It is parsed and interpreted, and subsequent arguments are consumed and formatted in an analogous manner to *printf*. *%s* takes the next argument and prints it in its default format. *%d* takes the next argument, which must be an *integral* type, and prints it in decimal. (Whether it is signed is determined by its type, not its format.) If the next argument is not an *integral* type, an exception is thrown.

Similarly, *%b*, *%o*, *%x*, and *%x* format *integral* types in binary, octal, lowercase hex, and uppercase hex, respectively. The %e, %E, %f, %F, %g, %G, %a, and %A take the next argument, which must be a floating-point type, and prints it in the corresponding floating format (they work the same as C99's floating-point formats).

And that's it for basic formats. No need for *l*, *ll*, *L*, or *hh* modifiers, or for *%u* and the like:

```
writef("%x %s", 123, 456L, 789);
                  // prints '7b 456789'
writef("%o ", 56, "%b", 22);
                  // prints '70 10110'
writef("%d %d %d", -1, -1, cast(uint)-1);
                  // prints -1 -1 4294967295
```

To the basic formats, *writef* adds in the flags, field width, and precision specifications wholesale from C99.

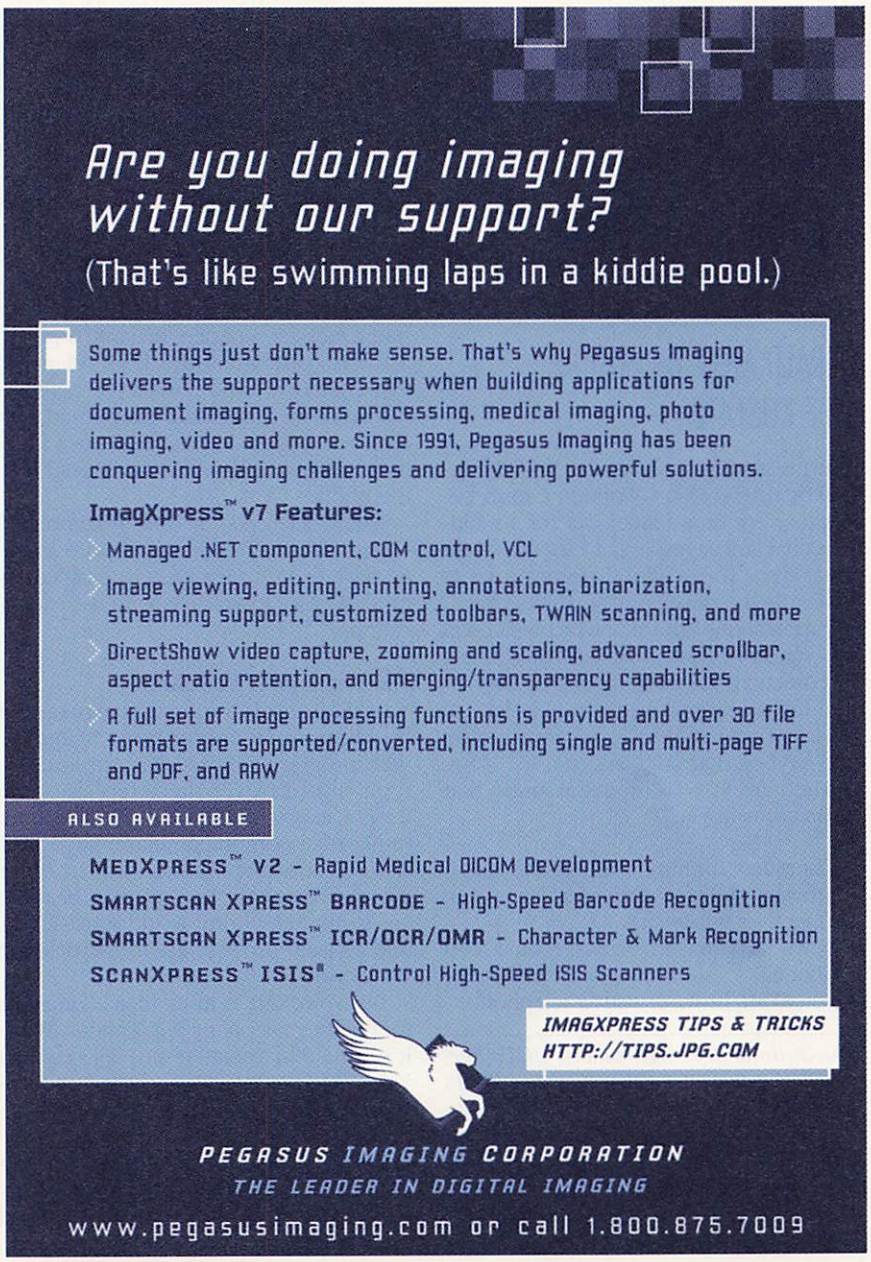
```
writef("%d|%-8d|%d", 123, 456, 789);
                  // prints '123|456|789'
```

So how does *writef* stack up against the complaints about *printf*? Well, *writef*:

- Is completely typesafe. Any unresolvable conflicts, such as trying to print a string in scientific notation, result in a runtime exception being thrown.
- Works for generic programming, because the format specifier no longer specifies the type. The type of the arguments are known to the formatter, so the programmer need not care what the underlying type of a typedef or template argument is.
- The format string only says how to print them, it does not contain type information.
- Can accept strings and characters in *char[]*, *wchar[]*, or *dchar[]* types corresponding to UTF-8, UTF-16, and UTF-32 character encodings. They can be mixed and matched even within one *writef*. There's no need to decide if the application is going to be *char* or *wchar_t*; each individual string can be whatever is most suitable to its data. *writef* can seamlessly combine them. Since UTF strings are an international standard, code using them is portable. The output of *writef* is converted to the character encoding used by the output stream.
- D arrays include length information, so attempts to overflow a buffer result in an exception being thrown rather than a crash or security breach.
- User-defined types are converted to strings via the user-defined *toString()* method.
- A small number of formats, coupled with intuitive defaults, means you'll be consulting the documentation a lot less frequently.

Recall the complaint that C++ *IOStreams* was neither exception safe nor thread safe with respect to the stream state. *writef* is exception safe because its formatting state is local on the stack, not globally set. It's thread safe when the implementation of *writef* synchronizes itself with *stdout* across the entire invocation of *writef*.

DDJ


Are you doing imaging without our support?
(That's like swimming laps in a kiddie pool.)
Some things just don't make sense. That's why Pegasus Imaging delivers the support necessary when building applications for document imaging, forms processing, medical imaging, photo imaging, video and more. Since 1991, Pegasus Imaging has been conquering imaging challenges and delivering powerful solutions.
ImagXpress™ v7 Features:
Managed .NET component, COM control, VCL
Image viewing, editing, printing, annotations, binarization, streaming support, customized toolbars, TWAIN scanning, and more
DirectShow video capture, zooming and scaling, advanced scrollbar, aspect ratio retention, and merging/transparency capabilities
A full set of image processing functions is provided and over 30 file formats are supported/converted, including single and multi-page TIFF and PDF, and RAW
ALSO AVAILABLE
MEDXPRESS™ V2 - Rapid Medical DICOM Development
SMARTSCAN XPRESS™ BARCODE - High-Speed Barcode Recognition
SMARTSCAN XPRESS™ ICR/OCR/OMR - Character & Mark Recognition
SCANXPRESS™ ISIS® - Control High-Speed ISIS Scanners
IMAGXPRESS TIPS & TRICKS
HTTP://TIPS.JPG.COM
PEGASUS IMAGING CORPORATION
THE LEADER IN DIGITAL IMAGING
www.pegasusimaging.com or call 1.800.875.7009

Adding Voice to XHTML

The X+V markup language is designed to do just that

GERALD MCCOBB AND JEFF KUSNITZ

In the spring of 2002, the W3C Multimodal Interaction Working Group began work on a framework for a multimodal language Standard for the World Wide Web. The goal of this Standard was to enable the development of interoperable applications that can interact with users in a variety of ways. Voice and digital pen, in particular, are new modes of interaction that may soon become popular, especially on small devices where a mouse and keyboard are difficult to use.

The current web model assumes that users are using a:

- Computer running a traditional visual browser (Internet Explorer, Navigator, Opera, or whatever).
- WML/WAP-based browser on a cell phone.
- VoiceXML browser with a telephone.

The authors are engineers in IBM's Pervasive Computing division. Gerald is IBM's secondary representative to the W3C Multimodal Working Group; and Jeff is representative to the VoiceXML Forum and the W3C Voice Browser working group. They can be contacted at mccobb@us.ibm.com and jk@us.ibm.com, respectively.

However, a new multimodal web model for applications supports the familiar user-interaction techniques (clicking, typing, tapping) along with additional modes such as speech recognition or text-to-speech. In such cases, users can tap, type, and talk, as well as see and hear. For example, users of a multimodal browser might pick up a phone, dial into a portal, ask "are there any flights from Atlanta to San Francisco," and have a list of flights displayed on the phone. They could then select one of the flights with either a stylus or voice and have the details read over the phone.

Jointly developed by IBM, Motorola, and Opera ASA, XHTML+Voice (X+V) is a multimodal markup language that enables this kind of voice interaction with web applications.

The V in X+V is VoiceXML

VoiceXML 2.0 is a markup language for building voice applications using the web-programming model. Applications typically run over the telephone and users use a telephone for input/output, rather than computers with a keyboard, mouse, and display monitor. In other words, VoiceXML and a voice browser are to the telephony world what HTML and a visual browser are to the desktop PC world.

But VoiceXML is more than a language for building telephony applications. While it's true that most, if not all, current VoiceXML implementations and applications are targeted at mobile users, the language is not strictly tied to a telephony interface. At its core, VoiceXML is a dialog markup language; it lets you easily build complex dialogs between users and computers.

X+V adds a modularized subset of VoiceXML to XHTML, as well as XML Events and a small number of element and attribute extensions to both XHTML and VoiceXML. The VoiceXML modules are defined according to the XHTML modularization framework and contain logical and hierarchical groupings of VoiceXML elements. For example, the executable content module contains the child elements of the *block*, *filled*, *catch*, *noinput*, *nomatch*, *error*, and *help* elements. Table 1 lists the VoiceXML modules and elements supported by X+V.

> "XHTML+Voice (X+V) is a multimodal markup language that enables voice interaction with web applications"

X+V Adds XML Events

XML Events is a W3C Recommendation for attaching event listeners and handlers

gdconf.com

GameDevelopers Conference

Feeling a bit lost on the open road?
Take control of your future today.

Thriving in today's interactive entertainment landscape requires developers to anticipate publisher needs and consumer desires to produce polished games with extremely high production quality. At GDC 2005, you will be equipped with practical tools to handle these and other obstacles along your road, and you will be exposed to future vision catalysts—empowering you to create tomorrow's game today.

Bring your future vision into focus at GDC 2005.

(continued from page 42)
to XML nodes. XML Events has a listener element and a set of attributes—event, handler, observer, target, and so on. They give authors the ability to observe an event as it flows through a node according to the DOM level 2 event model (supported by almost all web browsers and required by X+V), and activate a handler in response to that event. For example, a *click* event can be observed on an HTML input node, and a VoiceXML form can be activated in response to the click.

Whether HTML 4.01 intrinsic or VoiceXML events, all events in an X+V application flow through the XHTML tree. When users click on an HTML input, the *click* event flows from the *html* element to the input (the "capture" phase). Once it reaches the input node (the "target"), it flows back to the the *html* element (the "bubble" phase). The VoiceXML events processed by an active voice dialog are emitted to the XHTML container according to this rule: The XHTML node that activated the voice dialog is also where all the VoiceXML events emitted by the voice dialog can be observed; that is, the XHTML node is the "target" of the VoiceXML event. For example, if a voice dialog is activated in response to a *click* event on an input node, then while the voice dialog is active, a VoiceXML *help* event can be observed on the input node after users say "help."

X+V supports all HTML 4.01 intrinsic events, but removes the "on" prefix; "onload" becomes "load," for instance. VoiceXML events, on the other hand, are distinguished by the "vxml" prefix, so that "help" becomes "vxmlhelp" when it flows through the XHTML tree. There is also the *vxmldone* event, which is emitted by the VoiceXML form when it has successfully finished running.

Prototype the Voice Interface

Voice dialogs are defined within X+V as VoiceXML forms with unique IDs. Because voice dialogs are encapsulated by VoiceXML forms, the voice interactions for a multimodal application spanning multiple web pages can be prototyped separately from the visual HTML. Call-flow diagrams show the structure of the voice dialog in terms of statements, prompts and input gathering, internal processing, and decisions and gotos for branching to VoiceXML forms. Table 2 describes the operation of each object in a call-flow diagram and its mapping to VoiceXML.

The conditional and goto branching activate successive VoiceXML forms according to the logic of the application. The branching is implemented by X+V as one or more VoiceXML return elements. For each event (such as "vxml.goto.P00010") emitted by the VoiceXML form to the XHTML tree, there must be a corresponding listener that observes the event and activates the next VoiceXML form in response to the event:

```
<ev:listener ev:event="vxml.goto.P00010"
 ev:observer="bd1" ev:handler="#P00010"/>
```

The XML Events *listener* activates the VoiceXML form with ID *P00010* in response to the *vxml.goto.P00010* event, observed on the XHTML node with ID *bd1*.

The structure of the voice dialog is diagrammed by connecting the call-flow objects together, according to the convention that the call flow proceeds from left to right. Figure 1, the call-flow diagram for the sample X+V application we present here, represents a single web page. The *submit goto* object is a branch to a voice dialog associated with the next web page retrieved by the browser after the current page is submitted.

Add a Voice Dialog to XHTML

Once the structure of the voice dialog has been prototyped, the call-flow objects can be mapped to their VoiceXML representations; see the last column in Table 1. The VoiceXML form(s) can then be put into the head section of the XHTML page. The VoiceXML elements have to be given an arbitrary prefix (followed by a colon) that identifies them as belonging to the VoiceXML namespace. The specification of the prefix for the VoiceXML namespace

*Dynamic***PDF**™ components will revolutionize the way your websites and applications handle printable output. *Dynamic***PDF**™ components are available natively for .NET, Java and COM/ActiveX.

*Dynamic***PDF**™ Merger v3.0
Our popular and highly efficient class library for real time PDF document manipulation is available natively for .NET and Java
• 100% managed and safe code • Intuitive object model • Merging • Appending • Form-filling • Page placing, rotating and scaling • Splitting • Stamping • Seamless integration with Generator API

*Dynamic***PDF**™ Generator v3.0
Our flexible and highly efficient class library for real time PDF creation is available natively for .NET, Java and COM/ActiveX*
• Intuitive object model • 18 bar code symbologies • HTML Text • Unicode and CJK font support • Font embedding and subsetting • Flexible document templating • Custom page element API
Try our free community editions! *See our web site for COM product features

*Dynamic***PDF**™ ReportWriter v1.6
COM/ActiveX class library that integrates with ADO allowing for quick and easy generation of PDF reports
• Five section reports • Group-by functionality • Dynamic text sizing • Supports 18 reporting events • Font embedding • Layout mode • Custom format function • Page and section numbering

*Dynamic***PDF**™
www.dynamicpdf.com

ceTe software
INFINITE POSSIBILITIES
800.631.5006 +1 301.656.1331

is provided by an *xmlns* attribute added to the *html* element:

```
xmlns:vxml="http://www.w3.org/2001/vxml"
```

The next step is to add the X+V *sync* elements for connecting the voice input results to the XHTML input controls. The *sync* element synchronizes VoiceXML fields and XHTML input controls. For example, if users update a VoiceXML field using voice, the synchronized XHTML input is also updated with the results stored in the field. Conversely, if users update an XHTML input using the keyboard, the synchronized VoiceXML field is updated with the contents of the XHTML input.

The *sync* element belongs to the X+V namespace. The specification of the prefix for the X+V namespace is provided by an *xmlns* attribute added to the *html* element:

```
xmlns:xv="http://www.w3.org/2002/xhtml+
                                voice"
```

The sample X+V application we present here has two XHTML text inputs—*txtBoxDeptCity* for entering a departure city and *txtBoxDestCity* for an arrival city. The two text inputs are synchronized to the two VoiceXML fields, *fieldDeptCity* and *fieldDestCity*, by adding two X+V *sync* elements to the head section of the page:

```
<xv:sync xv:input="txtBoxDeptCity"
                xv:field="#fieldDeptCity"/>
<xv:sync xv:input="txtBoxDestCity"
                xv:field="#fieldDestCity"/>
```

The X+V *field* attribute references an ID placed on a VoiceXML field. Because VoiceXML does not include the *id* attribute with its field element, X+V adds the *xv:id* attribute that is referenced by the X+V *field* attribute:

```
<vxml:field xv:id="fieldDeptCity" name=
                              "fieldDeptCity">
```

All that is left is to activate the VoiceXML form in response to an XML Event. To activate the VoiceXML form in the sample X+V application when the page is loaded, the XML Events *event* and *handler* attributes are added to the XHTML *body* element:

```
<body id="bd1" ev:event="load"
                    ev:handler="runForm">
```

The *load* event observed on the body activates the VoiceXML form with ID *runForm*. The *event* and *handler* attributes are prefixed with an arbitrary label for the XML Events namespace. The specification of the prefix for the XML Events namespace provided by an *xmlns* attribute added to the *html* element is:

```
xmlns:ev="http://www.w3.org/2001/xmlevents"
```

Listing One presents a sample X+V application with the VoiceXML form and X+V *sync* elements added to the XHTML head section. Also included are script handlers for the *vxmlerror* and *vxmldone* events. Listing Two is the city grammar for the two VoiceXML fields. There are only seven major U.S. cities that can be matched by the grammar, but more cities can easily be added.

Running the Sample X+V Application

Opera (http://www.opera.com/) and Access (http://www.access-us-inc.com/) both have X+V browsers, which run on a number

Module	Purpose	Elements
Events	Catching and throwing VoiceXML events	catch, help, noinput, error, nomatch, throw
Executable Statements	Processing statements	assign, clear, var, reprompt, log
Filled	Processing when a field is filled by user input	filled
Flow Control	Conditional statements	if, else, elseif, return
Forms	Voice dialog forms and form items	form, field, block, record, initial, subdialog
Resources	Properties and parameters	property, param
Enumerate	Enumerate choices or options available to user	enumerate
Option	Input option for a field	option
Output	Voice and audio output	prompt, value, audio, SSML*
User Input	Voice input	grammar, SRGS**

Table 1: *VoiceXML modules supported by X+V. *Speech Synthesis Markup Language 1.0 (http://www.w3.org/TR/speech-synthesis/). **Speech Recognition Grammar Specification 1.0 (http://www.w3.org/TR/speech-grammar/).*

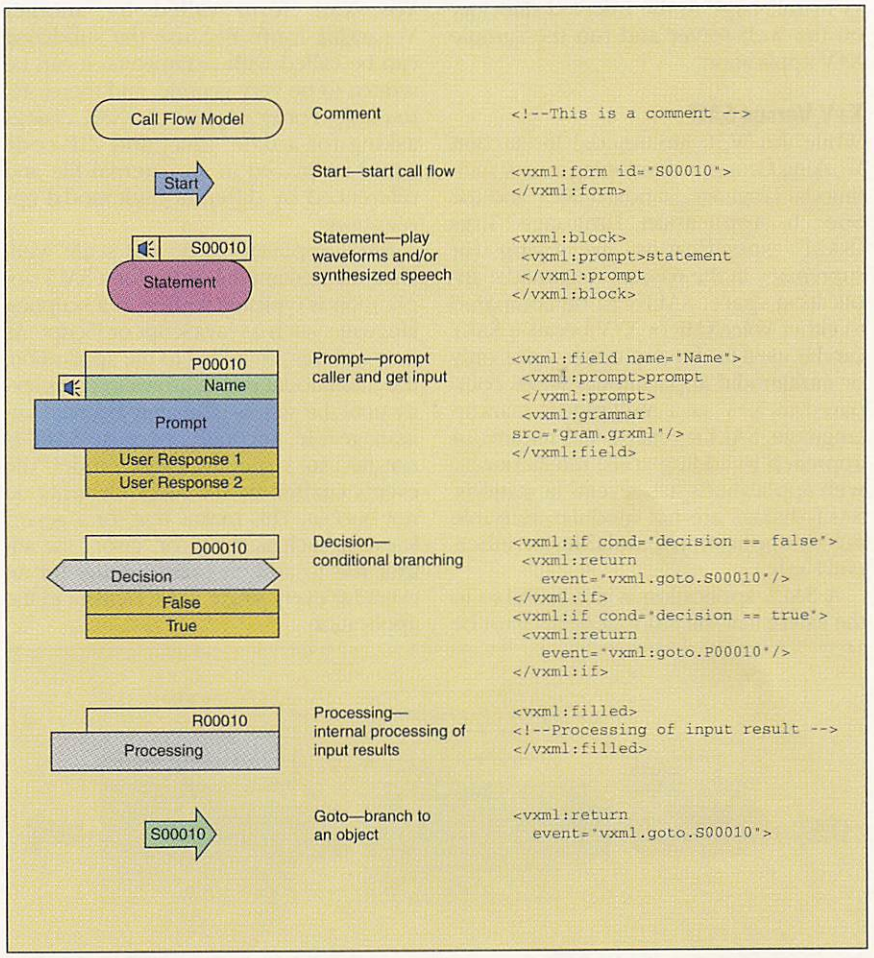


Table 2: *Call-flow objects with their descriptions and VoiceXML.*

of different client platforms, including embedded Linux, Pocket PC, and Windows. A free trial of the IBM multimodal tools (http://www-306.ibm.com/software/pervasive/multimodal/) includes Windows versions of both the Opera and Access Co.'s NetFront X+V browsers. Installed with the IBM multimodal toolkit are the embedded speech engines that the browsers need to run X+V applications.

The sample X+V application can be installed on any web server. Simply save the sample X+V markup in Listing One to a file called "sample-xv.mxml," to a directory that has been configured to deploy web applications. Also, save the file containing the city grammar, "city.grxml," to the same directory. Next, configure the content type the web server issues for files with the ".mxml" extension to "application/xhtml+voice+xml." For grammar files with the extension ".grxml," the content type is "application/srgs+xml." It is also a good idea to add the ".mxml" extension to the "text/html" content type, so that browsers that don't understand X+V can still run the application without voice interaction.

Start either the Opera or NetFront X+V browser and navigate to the "sample-xv.mxml" page in the installed directory on the web server and run the sample X+V application.

X+V Versus SALT

While the W3C Multimodal Interaction Working Group is developing a new multimodal language Standard, X+V and the Speech Application Language Tags (SALT) specification are vying for supremacy in the nascent multimodal application space. SALT can be compared to either VoiceXML or X+V because SALT can be used to develop either voice-only or multimodal applications. When compared to X+V as a multimodal markup language, SALT's generic and minimalist approach to adding voice interaction to web applications has several limitations. SALT dialogs are not modular, reusable across applications, scalable, or inherently safe.

A SALT application is not reusable to the extent that the SALT elements must be embedded in the application document. Every SALT application must be written from the ground up with a multimodal dialog dedicated to the application. However, macros that preprocess SALT source and tools that automatically generate SALT source may alleviate this problem. It is also true that SALT grammars are reusable as they can be referenced outside the current application document.

> "X+V is built upon the latest W3C Recommendations for visual interaction"

The VoiceXML dialogs in X+V can be referenced in an external file and thereby can be reused by the XHTML containers. X+V also supports the VoiceXML subdialog element. A subdialog is a VoiceXML form called by another VoiceXML form. Because the subdialog can be called with parameters, it can be written to be very generic, and therefore, reusable. For example, VoiceXML dialogs asking for a date, time, and ZIP code could be placed in an external file and referenced by different multimodal applications.

SALT applications do not scale well. First, the multimodal dialog for a SALT application is typically written in a scripting language, such as JavaScript or JScript. As new features are added to the application, the multimodal dialog grows in complexity until the scripts ultimately become unmanageable. Secondly, because SALT is not tied to a specific event model, the events emitted by the SALT elements do not bubble. This means that for a generic event, such as "help" or "error," the author has to add an event listener for an event for every single SALT element in the application.

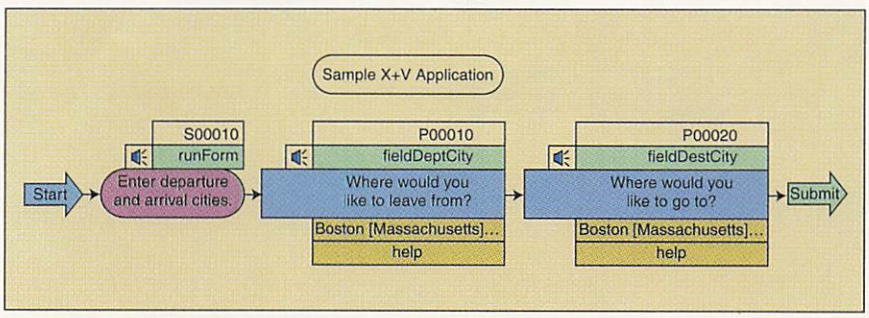


Figure 1: *Call-flow diagram.*

According to the SALT 1.0 specification, the "point is to show that all possible user input and error events are caught and safely handled, so that the dialog is never left in a 'hanging' state." This implies that SALT developers must be careful or the application will not work properly, and all error events must be explicitly handled. VoiceXML dialogs, on the other hand, are inherently "safe": All error conditions are handled even if the author chooses not to handle them.

Conclusion

Because X+V's voice dialogs are modular, X+V lets you prototype the voice interface separately from the visual interface. This is important because the voice interface is as important as the visual interface. For example, multimodal applications can also be voice-only or hands-free. Separating the voice interface makes building an X+V multimodal application straightforward:

1. Develop the call-flow of the voice interaction with the application.
2. Create the VoiceXML dialogs (or reuse if possible) and XML Event listeners from the call-flow diagram (or generate them if a tool is available).
3. Create the visual HTML elements of the application (or reuse if it is a legacy application).
4. Drop the VoiceXML forms and XML Events listeners into one or more web pages of the application.
5. Add the X+V <*sync*> tags to the web pages to synchronize VoiceXML field elements with the HTML visual controls.

X+V is built upon the latest W3C Recommendations for visual interaction (XHTML 1.1), authoring event listeners and handlers (XML Events), and voice interaction (VoiceXML 2.0). While both SALT and X+V make use of many of the same web Standards—Speech Synthesis Markup Language (SSML) and Speech Recognition Grammar Specification (SRGS), for instance—only X+V has chartered a path that will continue to align with the W3C as its Standards evolve.

Resources

- XHTML+Voice 1.2 Specification (http://www.voicexml.org/specs/multimodal/x+v/12/).
- Speech Application Language Tags (SALT) 1.0 Specification (http://www.saltforum.org/saltforum/downloads/SALT1.0.pdf).
- Voice Extensible Markup Language (VoiceXML) Version 2.0 (http://www.w3.org/TR/voicexml20/).
- XHTML 1.1 (http://www.w3.org/TR/xhtml11/).

- Speech Synthesis Markup Language (SSML) Version 1.0 (http://www.w3.org/TR/speech-synthesis/).
- Speech Recognition Grammar Specification (SRGS) Version 1.0 (http://www.w3.org/TR/speech-grammar/).
- XML Events (http://www.w3.org/TR/xml-events/).
- Semantic Interpretation (http://www.w3.org/TR/semantic-interpretation/).
- W3C World Wide Web Consortium (http://www.w3.org/).
- W3C Multimodal Interaction Working Group (MMI WG) (http://www.w3.org/2002/mmi/).

DDJ

Listing One

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:vxml="http://www.w3.org/2001/vxml"
      xmlns:ev="http://www.w3.org/2001/xml-events"
      xmlns:xv="http://www.voicexml.org/2002/xhtml+voice">
  <head><title>Sample X+V Application</title>
    <!-- VoiceXML -->
    <vxml:form id="RunForm">
      <vxml:block>Enter departure and arrival cities</vxml:block>
      <vxml:field xv:id="fieldDeptCity" name="fieldDeptCity">
        <vxml:grammar src="city.grxml"/>
        <vxml:prompt>Where would you like to leave from?
        </vxml:prompt>
      </vxml:field>
      <vxml:field xv:id="fieldDestCity" name="fieldDestCity">
        <vxml:grammar src="city.grxml"/>
        <vxml:prompt>Where would you like to go?</vxml:prompt>
      </vxml:field>
      <vxml:catch event="nomatch noinput help">
          For example, say New York.
      </vxml:catch>
    </vxml:form>
    <!-- sync's -->
    <xv:sync xv:input="txtBoxDeptCity" xv:field="#fieldDeptCity"/>
    <xv:sync xv:input="txtBoxDestCity" xv:field="#fieldDestCity"/>
    <!-- scripts -->
    <script ev:event="vxmlerror" ev:observer="bd1" declare="declare">
    document.getElementById("pg1").innerHTML =
                        "Sorry, there was a speech error";
    </script>
    <script ev:event="vxmldone" ev:observer="bd1" declare="declare">
        document.getElementById("pg1").innerHTML = "Departure city: "
             + document.travelForm.txtBoxDeptCity.value
             + "  Arrival city: "
             + document.travelForm.txtBoxDestCity.value + ".";
    </script>
  </head>
  <body id="bd1" ev:event="load" ev:handler="#RunForm">
    <p id="pg1">Enter departure and arrival cities</p>
    <form name="travelForm" action=".">
      <input name="txtBoxDeptCity" type="text" /> 
      <input name="txtBoxDestCity" type="text" />
    </form>
  </body>
</html>
```

Listing Two

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//EN"
                  "http://www.w3.org/TR/speech-grammar/grammar.dtd">
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar"
         tag-format="semantics/1.0"
         mode="voice" xml:lang="en-US" root="uscity">
<rule id="california">
    <one-of>
        <item>
            <one-of>
                <item> Los Angeles </item>
                <item> L.A. </item>
            </one-of><tag><![CDATA[$="Los Angeles"]]></tag>
        </item>
        <item> San Francisco
             <tag><![CDATA[$="San Francisco"]]></tag>
        </item>
    </one-of>
</rule>
<rule id="florida">
    <one-of>
        <item> Miami <tag><![CDATA[$="Miami"]]></tag></item>
    </one-of>
</rule>
<rule id="illinois">
    <one-of>
        <item>Chicago<tag><![CDATA[$="Chicago"]]></tag></item>
    </one-of>
</rule>
<rule id="massachusetts">
    <one-of>
        <item>Boston <tag><![CDATA[$="Boston"]]></tag></item>
    </one-of>
</rule>
<rule id="newyork">
    <one-of>
        <item> New York
            <tag><![CDATA[$="New York City"]]></tag>
        </item>
    </one-of>
</rule>
<rule id="washingtonDC">
    <one-of>
        <item> Washington </item>
        <item>
            <item repeat="0-1"> Washington </item> D.C.
        </item>
    </one-of><tag><![CDATA[$="District of Columbia"]]></tag>
</rule>
<rule id="uscity" scope="public">
    <one-of>
        <item>
          <ruleref uri="#california"/>
             <tag><![CDATA[ $= $california;]]></tag></item>
        <item><ruleref uri="#florida"/>
             <tag><![CDATA[ $= $florida;]]></tag></item>
        <item><ruleref uri="#illinois"/>
             <tag><![CDATA[ $= $illinois;]]></tag></item>
        <item><ruleref uri="#massachusetts"/>
             <tag><![CDATA[ $= $massachusetts;]]></tag></item>
        <item><ruleref uri="#newyork"/>
             <tag><![CDATA[ $= $newyork;]]></tag></item>
        <item><ruleref uri="#washingtonDC"/>
             <tag><![CDATA[ $= $washingtonDC;]]></tag></item>
    </one-of>
</rule>
</grammar>
```

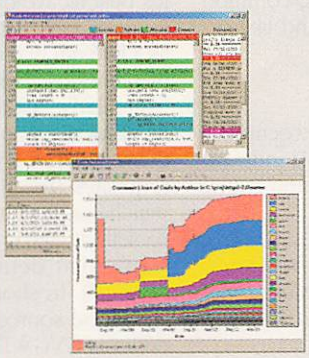
DDJ

Off-Shore Development Management Tools

- Anticipate problems through trend analysis
- Track developer activity
- Review code remotely
- Automate audit trails
- Integrate with version control software

Outsourcing Giving you Headaches?

Get the insight you need to deliver quality software on time.



Take back control with the SmartBear software suite:

CodeReports™
CodeReviewer™
CodeMetrics™
CodeHistorian™

SmartBear Software™
http://smartbearsoftware.com
info@ smartbearsoftware.com
877.501.5651

Cross-Platform Builds

All it takes is five rules and 42 lines

JOHN GRAHAM-CUMMING

Programmers often find themselves producing cross-platform software, which requires writing and building code that works on Linux, Solaris, HP-UX, AIX, and Windows. And whether you're a professional build manager or simply a programmer charged with maintaining the Makefiles, you know that multiple platforms means multiple Make hassles. In this article, I present a cross-platform pattern for Makefiles that works across Windows and common UNIX platforms, minimizing platform-specific code. At the same time, the pattern completely eliminates recursive use of Make, another hassle that complicates a build manager's life. To simplify the discussion, I've stuck to compiling C code, but there's nothing in the technique that excludes languages such as C++, C#, or Java.

For the purposes of illustration, imagine this layout of directories and source code:

```
src
src/library
src/executable
```

Further imagine a library (lib.a on UNIX, lib.lib on Windows) containing common code built from C files lib1.c and lib2.c:

```
src/library/lib1.c
src/library/lib2.c
```

And finally, imagine the executable (exec on UNIX, exec.exe on Windows) compiled from foo.c and bar.c and linked with the library.

John is chief scientist at Electric Cloud, which focuses on reducing software build times. He can be contacted at jgc@electric-cloud.com.

```
src/executable/foo.c
src/executable/bar.c
```

Rule #1: Use GNU Make

The GNU Project's GNU Make (gmake) is powerful, has broad platform support, and has excellent documentation. It simplifies the task of building a cross-platform Make system if you follow the rest of the rules in this article. (Use GNU Make 3.80 or higher since I rely here on some recent features; see http://www.gnu.org/software/make/.)

The following is the top-level Makefile stored in src/:

```
MODULES=library executable
include $(addsuffix /Makefile,$(MODULES))
```

Typing *make* in src/ builds all the object files, library, and executable. The *MODULES* variable is a list of the directories containing components that must be built. *include $(addsuffix /Makefile,$(MODULES))* turns into *include library/Makefile executable/Makefile* and includes the Makefiles that describe each of the separate components.

Rule #2: No Spaces in Filenames, No Drive Letters

GNU Make (in common with other Make programs and scripting languages) treats a space as a list separator. (You can see that in the top-level Makefile where *MODULES* is a list of directory names and *$(addsuffix)* iterates over that list.) Because of this, it's best to avoid spaces in filenames (for instance, GNU Make sees C:\Program Files as a list with two elements). Either use short filenames or avoid spaces altogether.

Similarly, Make provides special treatment for the colon (:), typically using it as the separator between a target and its prerequisites (foo: foo.o bar.o, and so on), which can lead to confusion if Make sees something like *c:\foo: foo.o bar.o*. ("Is this a rule that defines how to make *c* or *c:\foo*?"). In short, use relative paths.

A good source of GNU Make on Windows is CYGWIN. CYGWIN provides a UNIX-style pathname for drive letters. In a CYGWIN shell, */cygdrive/c* is *C:*. Use CYGWIN paths if you must have access to a specific drive.

The Makefile in the library/ subdirectory consists of five lines:

```
include top.mak
SRCS = lib1.c lib2.c
BINARY = lib
BINARY_EXT=$(X_LIBEXT)
DEPS=
include bottom.mak
```

> "GNU Make does a good job of tracking the dependencies between files"

Evidently, all the real work is being done by top.mak and bottom.mak.

Rule #3: Decide What Platform the Make is Running On

The simplest way to determine your platform is the *uname* command (available on all flavors of UNIX and on Windows under CYGWIN). The *uname* options *m* and *s* give the processor type and the operating system. For example, on a Linux-based PC, *uname -ms* returns Linux i686. One of the things top.mak does is get the architecture into a variable called "*X_ARCH*":

```
X_ARCH := $(shell uname -ms | sed -e s"/ /_/g")
```

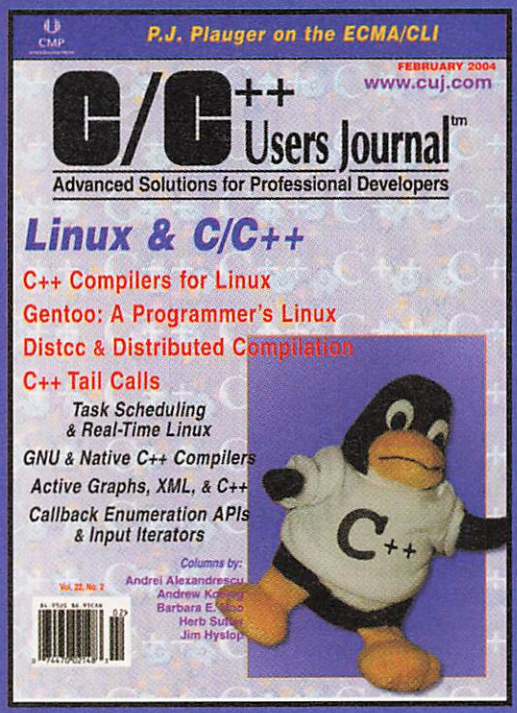
The *sed* invocation turns any spaces in the output from *uname* into underscores. So, on CYGWIN, *X_ARCH* will be *CYGWIN_NT-5.0_i686*; on Solaris, *SunOS_sun4u*; and on Linux, *Linux_i686*. top.mak

BUILD WITH THE BEST!

C/C++ Users Journal gives you the tightest, fastest code possible from experts in C/C++ programming.

Every month you'll get features such as:

- Internet Programming
- Object-Oriented Programming
- Embedded Systems
- Graphics
- C# and .NET
- And more...



Subscribe Today 12 issues for $19.95

C/C++ Users Journal™

Advanced Solutions for Professional Developers

VISIT US AT: www.cuj.com/sub/

Discount KeyCode 2CDT

Canada/Mexico Subscriptions: 1 year - $40 / Outside North America: 1 year - $60

CMP
United Business Media

PUT POWER IN YOUR HANDS!

(continued from page 48)
begins by determining the machine architecture, setting *X_ARCH*:

```
ifndef X_ALREADYLOADED
.PHONY: all
all::
X_ARCH := $(shell uname -ms | sed -e s"/ /_/g")
endif
```

Notice how the definition of *X_ARCH* uses the GNU Make := operator. This causes the RHS to be evaluated at once and the result placed in *X_ARCH*. If the more familiar = operator were used, every reference to *X_ARCH* (and there could be hundreds because of the recursive nature of Make macros) would require a shell invocation to execute *uname* and *sed*, wasting valuable time in the process.

This code also defines a phony target (that is, a target that isn't a file) called *all*. Since this is the first target mentioned, if you just type *make*, it builds the *all* target. *all* is defined using the :: syntax, which lets you later add other commands and prerequisites as each individual Makefile is parsed. You use this to define all for each of the modules.

ifndef X_ALREADYLOADED ensures that this block of code is only parsed once (later in top.mak, you set *X_ALREADYLOADED* to 1).

Rule #4: Separate Output on a Per-Platform Basis

GNU Make does a good job of tracking the dependencies between files, but cross-platform Make and network mounted directories mean that it would be easy to confuse GNU Make. If the object files being created aren't separated on a per-platform basis, an *.o* built on Solaris could clash with one built on Linux, and so on.

top.mak strictly separates the output by platform (and by module) by defining three variables used elsewhere in the system (throughout top.mak and bottom.mak, variables are prefixed by *X_* to stop them from clashing with variables you might want to use in the rest of your system):

- *X_OUTARCH*, the location where objects for this architecture should be placed (by default, this is a subdirectory of src/; on PC Linux it would be src/Linux_i686).
- *X_MODULE*, the name of the module being built. This is the name of a subdirectory of src/ (library or executable).
- A specific variable for the module being built giving the location of its object files. For the library module, library_OUTPUT is src/Linux_i686/library and executable_OUTPUT is src/Linux_i686/executable.

top.mak starts by setting *X_OUTARCH*, which by default is a subdirectory of the current working directory (src/, for example), but can be overridden by setting *X_OUTTOP* on the command-line of GNU Make (or in the environment):

```
ifndef X_ALREADYLOADED
X_OUTTOP ?= .
X_OUTARCH := $(X_OUTTOP)/$(X_ARCH)
endif
```

top.mak then sets *X_MODULE*. To do this it uses a GNU Make 3.80 feature—the variable *MAKEFILE_LIST*, which is a (space-separated) list of the Makefiles included so far. By stripping out references to top.mak and bottom.mak, you set the *X_MAKEFILES* variable, the last element of which will be the Makefile, which included top.mak; for instance, the Makefile to one of the modules. The complex definition of *X_MODULE* first takes the last element of *X_MAKEFILES* using *$(word)* (library/Makefile), gets just the directory portion using *$(dir)* (library/), and strips the trailing / with *$(patsubst)*.

```
X_MAKEFILES :=
    $(filter-out %.mak,$(MAKEFILE_LIST))
X_MODULE :=
    $(patsubst %/,%,$(dir $(word $(words
$(X_MAKEFILES)),$(X_MAKEFILES))))
```

The next step requires a little mind-bending—you define a variable with a computed name. The variable name is *$(X_MODULE)_OUTPUT* whose name is determined by the value of *X_MODULE* at the time the variable is defined (that is, either library or executable).

```
$(X_MODULE)_OUTPUT :=
        $(X_OUTARCH)/$(X_MODULE)
```

This variable contains the name of the directory into which the module's object files will be written. To ensure that the directory exists, you set a dummy variable to the output of a call to mkdir using GNU Make's *$(shell)* command and you use the := operator to perform the command at once. Notice how you use the value of *$(X_MODULE)_OUTPUT*, which requires writing *$($(X_MODULE)_OUTPUT)*:

```
X_IGNORE :=
  $(shell mkdir -p $($(X_MODULE)_OUTPUT))
```

Lastly, top.mak defines a clean target that uses the fact that you know that the output for this architecture is in the directory named in *X_OUTARCH*. A single *rm -rf* cleans up all objects:

```
ifndef X_ALREADYLOADED
X_ALREADYLOADED = 1
.PHONY: clean
clean::
  @rm -rf $(X_OUTARCH)
.SUFFIXES:
endif
```

You'll notice that the line *.SUFFIXES:* is included in the code, which brings me to the final rule.

Rule #5: Override Built-In Rules

GNU Make (and other Makes) include extensive built-in rules describing how to build objects from various source types (.o from .c, .obj from .cpp, and so on). It's best to override them for two reasons:

- Searching all the possible ways of making *X* from *Y* (because the rules can chain together) can be slow.
- You want per-platform control so that you can specify flags, system libraries, and file locations.

Writing *.SUFFIXES:* turns off GNU Make's built-in rules, which brings you to bottom.mak, where the rules are defined. Recall the Makefile in the library/ directory:

```
include top.mak
SRCS = lib1.c lib2.c
BINARY = lib
BINARY_EXT=$(X_LIBEXT)
DEPS=
include bottom.mak
```

As GNU Make proceeds through parsing, it includes top.mak (and thus sets *X_MODULE, X_ARCH*, and *$(X_MODULE)_OUTPUT)*. Furthermore, the Makefile sets the following:

- *SRCS*, the list of source files used to build this module.
- *BINARY*, the name of the binary (minus extension) that this module creates.
- *BINARY_EXT*, the extension for the binary. Since this is platform specific, you reference yet another variable *X_LIBEXT*, which gives the platform-specific extension for a library (.a or .lib). *X_LIBEXT* (*X_EXEEXT* for executables) and *X_OBJEXT* (for regular object files) are set by a platform-specific Makefile that bottom.mak includes. For example, bottom.mak includes Linux_i686.mak on PC Linux that specifies (among other things):

```
X_OBJEXT=.o
X_LIBEXT=.a
X_EXEEXT=
```

bottom.mak does the final work of setting up the dependencies for the module and the rules to build the objects, library, or executable. First, it takes the lists of source files and transforms them into a list of object files in the right output directory (module and architecture specific), then it does the same with the binary.

```
include $(X_ARCH).mak
$(X_MODULE)_OBJS =
    $(addsuffix $(X_OBJEXT),$(addprefix
$($(X_MODULE)_OUTPUT)/,
    $(basename $(SRCS)))) $(DEPS)
```

```
$(X_MODULE)_BINARY =
    $(addprefix $($(X_MODULE)_OUTPUT)/,
    $(BINARY))$(BINARY_EXT)
include $(X_ARCH)-rules.mak
```

Once again, the variable names are computed so that the same macro definitions can be written once and create separate variables for each module: *$(X_MODULE)_OBJS* (*library_OBJS* for the library module) is the list of object files for the module (Linux_i686/library/lib1.o Linux_i686/library/lib2.o), and *$(X_MODULE)_BINARY* (library_BINARY) is the name of the binary (Linux_i686/library/lib.a).

Then you specify a couple of rules. The first says that the *all* target builds the binary. The second says that, if users specify the name of the module, then it also builds the binary. This means that *all* will build every binary, although it's possible to use *make library* to build the library module.

```
all:: $($(X_MODULE)_BINARY)
$(X_MODULE): $($(X_MODULE)_BINARY)
```

You'll notice that bottom.mak included a platform-specific Makefile with the directives *include $(X_ARCH).mak* and *include $(X_ARCH)-rules.mak*, which contain all the platform code you need. Here's *Linux_i686.mak*:

```
X_OBJEXT=.o
X_LIBEXT=.a
X_EXEEXT=
```

and here's *Linux_i686-rules.mak*:

```
$($(X_MODULE)_OUTPUT)/%.o:
        $(X_MODULE)/%.c
    @$(COMPILE.c) -o '$@' '$<'
$($(X_MODULE)_OUTPUT)/$(BINARY).a:
        $($(X_MODULE)_OBJS)
    @$(AR) r '$@' $^
    @ranlib '$@'
$($(X_MODULE)_OUTPUT)/
        $(BINARY)$(X_EXEEXT):
$($(X_MODULE)_OBJS)
    @$(LINK.c) $^ -o'$@'
```

These two files first specify the extensions that are valid for the platform and then define rules for building an *.o* from a *.c*, a *.a* from the list of objects for a module, and an executable from a list of objects. Notice how each rule uses *$($(X_MODULE) _OUTPUT)* for the architecture- and module-specific output location for binaries. The rules are using GNU Make built-in variables COMPILE.c (how to compile a C file), AR (ar), and LINK.c (how to link). If necessary, these can be overridden to have platform-specific flags and other options.

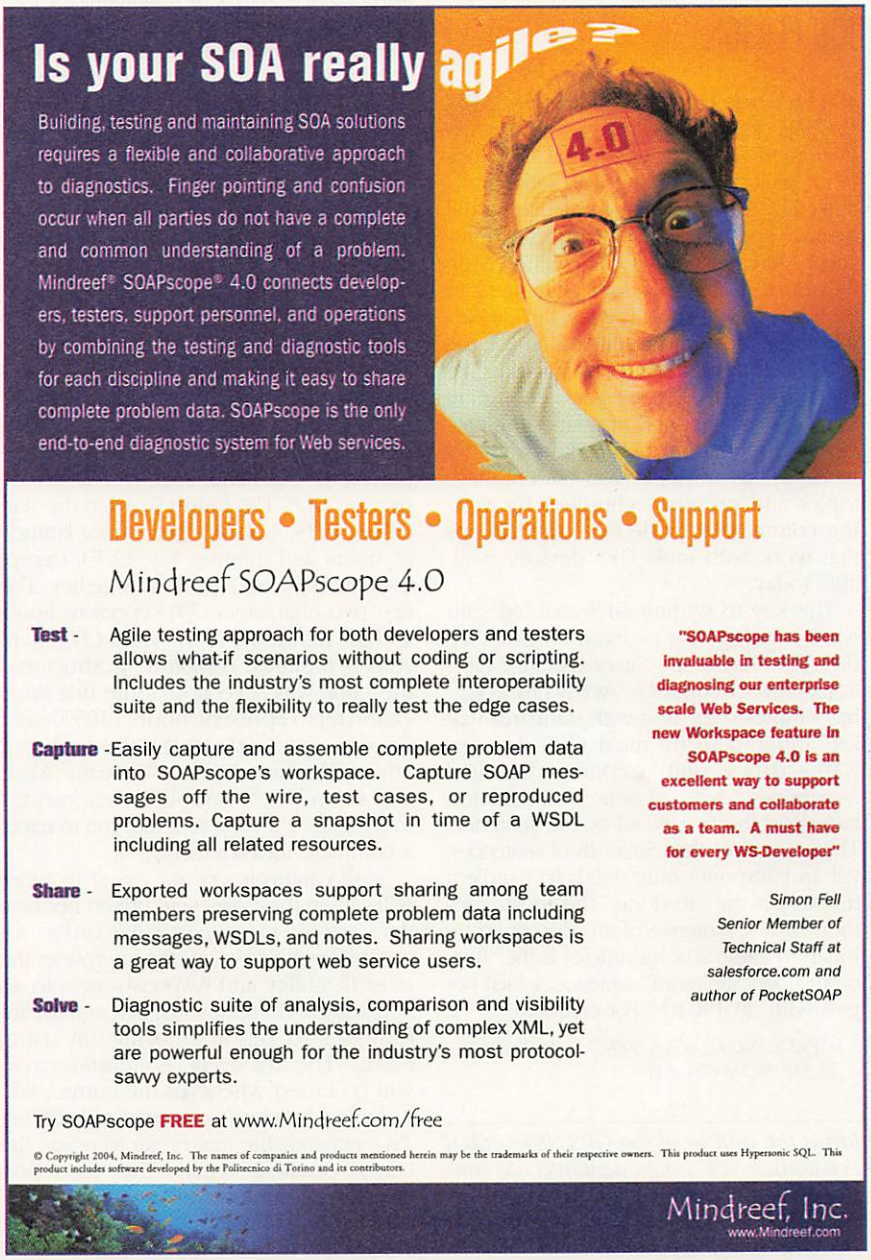
The only change needed to build the same code on another platform is the creation of the appropriate equivalents of *Linux_i686.mak* and *Linux_i686-rules.mak*. For example, under Windows the extensions for objects, libraries, and executables are .obj, .lib, and .exe and the rules can be redefined to use Microsoft's cl compiler (and done so without making any changes to the Makefile in the module directories or top.mak and bottom.mak).

If you're a Make guru, you're probably ready to point out that the executable needs to link against the library and needs to depend on the library itself. Here's the Makefile used in the executable/directory:

```
include top.mak
SRCS = foo.c bar.c
BINARY = exec
BINARY_EXT=$(X_EXEEXT)
DEPS = $(library_BINARY)
include bottom.mak
```

The interesting line is *DEPS=$(library_BINARY)*, which specifies that this module depends on the binary file generated by the library module. Because the system has set up separate variables for each module, the executable module can refer directly to the binary or objects created by another module. All that matters is that the Makefile for library was included before the Makefile for executable. Put together, top.mak, bottom.mak, and the platform-specific Makefiles consist of 42 lines of Make syntax that gives a powerful, flexible cross-platform Make system that is fast and eliminates the problems inherent in recursive Makes.

DDJ


Is your SOA really agile?
Building, testing and maintaining SOA solutions requires a flexible and collaborative approach to diagnostics. Finger pointing and confusion occur when all parties do not have a complete and common understanding of a problem. Mindreef® SOAPscope® 4.0 connects developers, testers, support personnel, and operations by combining the testing and diagnostic tools for each discipline and making it easy to share complete problem data. SOAPscope is the only end-to-end diagnostic system for Web services.
4.0
Developers • Testers • Operations • Support
Mindreef SOAPscope 4.0
Test - Agile testing approach for both developers and testers allows what-if scenarios without coding or scripting. Includes the industry's most complete interoperability suite and the flexibility to really test the edge cases.
Capture - Easily capture and assemble complete problem data into SOAPscope's workspace. Capture SOAP messages off the wire, test cases, or reproduced problems. Capture a snapshot in time of a WSDL including all related resources.
Share - Exported workspaces support sharing among team members preserving complete problem data including messages, WSDLs, and notes. Sharing workspaces is a great way to support web service users.
Solve - Diagnostic suite of analysis, comparison and visibility tools simplifies the understanding of complex XML, yet are powerful enough for the industry's most protocol-savvy experts.
"SOAPscope has been invaluable in testing and diagnosing our enterprise scale Web Services. The new Workspace feature in SOAPscope 4.0 is an excellent way to support customers and collaborate as a team. A must have for every WS-Developer"
Simon Fell
Senior Member of Technical Staff at salesforce.com and author of PocketSOAP
Try SOAPscope FREE at www.Mindreef.com/free
© Copyright 2004, Mindreef, Inc. The names of companies and products mentioned herein may be the trademarks of their respective owners. This product uses Hypersonic SQL. This product includes software developed by the Politecnico di Torino and its contributors.
Mindreef, Inc.
www.Mindreef.com

Writing GPS Applications

Accurately interpreting satellite data is half the job

JON PERSON

In his article "GPS Programming & .NET" (*DDJ*, June 2004), Johan Franson examined several fundamental GPS topics, including working with serial ports, detecting data errors, and calculating the distance and bearing with Universal Transverse Mercator (UTM) coordinates. Still, there are more questions that need to be addressed, especially for developers who want to write applications for the real world. For instance, what do GPS applications need to be good enough at to pilot airplanes? And how does the process of interpreting GPS data actually work? In this article, I cover both of these topics and present techniques for writing commercial-grade GPS applications that work with most GPS devices available today.

The key to writing GPS-enabled software is being able to interpret raw GPS data. Fortunately, the National Marine Electronics Association (http://www.nmea.org/) has simplified the task with standards that are adhered to by most GPS devices. NMEA data is sent as comma-delimited "sentences," which contain information based on the first word of the sentence. There are more than 50 kinds of sentences, yet an interpreter only needs to handle a few to get the job done. The most common NMEA sentence of all (described by Johan in Table 2 of his article) is the "Recommended Minimum" sentence, which begins with "$GPRMC." For example:

```
$GPRMC,040302.663,A,3939.7,N,10506.6,W,0
.27,358.86,200804,,*1F
```

Jon is the author of the GPS.NET Global Positioning SDK component and has consulted for military companies and the Forest Service on GPS solutions. He can be reached at info@gpsdotnet.com.

This single sentence contains nearly everything GPS applications need—latitude, longitude, speed, bearing, satellite-derived time, fix status, and magnetic variation.

The Core of an Interpreter

The first step in interpreting raw NMEA data is to build or otherwise utilize an interpreter. To give you a headstart, I present source code from my GPS.NET Global Positioning SDK component (http://www.gpsdotnet.com/). The code is stripped of features such as multithreading and error handling for brevity. The interpreter does two things: separates each sentence into its individual words and examines the first word to figure out what information is available to extract. Listing One is a function that's the start of the interpreter class.

The next step is to perform actual extraction of information, starting with latitude and longitude. Latitude and longitude are stored in the form "DDD°MM'SS.S," where *D* represents hours (also called "degrees"), *M* represents minutes, and *S* represents seconds. Coordinates can be displayed in shorthand, such as *DD°MM.M* or even *DD°*. The fourth word in the sentence, *3939.7*, shows the current latitude as hours and minutes (39°39.7'), except the numbers are squished together. The first two characters (39) represent hours and the remainder of the word (39.7) represents minutes. Longitude is structured the same way, except that the first three characters represent hours (105°06.6'). Words 5 and 7 indicate the "hemisphere," where *N* means North, *W* means West, and so on. The hemisphere is appended to the end of the numeric portion to make a complete measurement.

NMEA interpreters are easier to work with when they are event driven because data arrives in no particular order. An event-driven class gives the interpreter the most flexibility and responsiveness to an application. Consequently, I design the interpreter to report information using events. The first event, *PositionReceived*, will be raised whenever the current latitude and longitude are received. Listing Two expands the interpreter to report the current position, so that it then can report the current latitude and longitude.

One thing to watch out for here is that some GPS devices report blank values when no information is known. Therefore, it's a good idea to test each word for a value before parsing. If you need to type the degree symbol (°), hold down the Alt key and type "0176" on the numeric keypad.

Taking Out the Garbage

Each sentence is accompanied by a checksum used to detect any errors in NMEA data. A checksum is calculated as the XOR of bytes between (but not including) the dollar sign ($) and asterisk (*). This checksum is then compared with the checksum from the sentence. If the checksums do not match, the sentence is typically discarded. This is okay to do because the GPS devices tend to repeat the same information every few seconds. I modified Johan's *ValidateChecksum* method slightly to divide the checksum calculation and comparison procedures, but the technique is still the same. With the ability to compare checksums, the interpreter is able to throw out any sentence with an invalid checksum. Listing Three expands the interpreter to detect errors and parse only error-free NMEA data.

"NMEA interpreters are easier to work with when they are event driven"

Wireless Atomic Time

Time is the cornerstone of GPS technology because distances are measured at the speed of light. Each GPS satellite contains four atomic clocks that it uses to time its radio transmissions within a few nanoseconds. One fascinating feature is that with just a few lines of code, these atomic clocks can be used to synchronize a computer's clock with millisecond accuracy. The

second word of the $GPRMC sentence—040302.663— contains satellite-derived time in a compressed format. The first two characters represent hours, the next two represent minutes, the next two represent seconds, and everything after the decimal place is milliseconds. So, the time is 4:03:02.663 AM. However, satellites report time in universal time (GMT+0), so the time must to be adjusted to the local time zone. Listing Four (available electronically; see "Resource Center," page 5) adds support for satellite-derived time and uses the *DateTime.ToLocalTime* method to convert satellite time to the local time zone.

Direction and Speed Alerts

GPS devices analyze your position over time to calculate speed and bearing. The $GPRMC sentence previously presented also includes these readings. Speed is always reported in knots and bearing is reported as an "azimuth," a measurement around the horizon measured clockwise from 0° to 360° where 0° represents north, 90° means east, and so on. A little math is applied to convert knots into miles per hour. The power of GPS is again demonstrated with one line of code in Listing Five (available electronically) that figures out if a car is over the speed limit.

Are We Fixed Yet?

The $GPRMC sentence also includes a value that indicates whether a "fix" has been obtained. A fix is possible when the signal strength of at least three satellites is strong enough to be involved in calculating your location. If at least four satellites are involved, altitude also becomes known. The third word of the $GPRMC sentence is one of two letters: "A" for "active," where a fix is obtained, or "V" for "invalid" where no fix is present. Listing Six (available electronically) includes code to examine this character and report on the fix status.

As you can see, a lot of information is packed into a single NMEA sentence. Now that the $GPRMC sentence has been fully interpreted, the interpreter can be expanded to support a second sentence—$GPGSV. This sentence describes the configuration of satellites overhead, in real time.

Real-Time Satellite Tracking

Knowing the location of satellites is important when determining how precise readings are and how stable a GPS fix is. I address GPS precision later in this article. Here, I focus on interpreting satellite location and signal strength.

There are 24 operational satellites in orbit. Satellites are spaced in orbit so that at any time, a minimum of six satellites will be in view to users anywhere in the world. Satellites are constantly in motion, which is good because it prevents the existence of "blind spots" in the world with little or no satellite visibility. Just like finding stars in the sky, satellite locations are described as the combination of an azimuth and an elevation. Again, azimuth measures a direction around the horizon. Elevation measures a degree value up from the horizon between 0° and 90°, where 0° represents the horizon and 90° represents "zenith," directly overhead. So, if the device says a satellite's azimuth is 45° and its elevation is 45°, the satellite is located halfway up from the horizon towards the northeast. In addition to location, devices report each satellite's Pseudo-Random Code (PRC), which is a number used to uniquely identify one satellite from another.

Here's an example of a $GPGSV sentence:

```
$GPGSV,3,1,10,24,82,023,40,05,62,285,32,01,
62,123,00,17,59,229,28*70
```

Each sentence contains up to four blocks of satellite information, comprised of four words. For example, the first block is "24,82,023,40," the second block is "05,62,285,32," and so on. The first word of each block gives the satellite's PRC. The second word gives each satellite's elevation, followed by azimuth and signal strength. If this satellite information were to be shown graphically, it looks like Figure 1.

Perhaps the most important number in this sentence is the signal-to-noise ratio (SNR). This number indicates how strongly a satellite's radio signal is being received. Remember, satellites transmit signals at the same strength but things, such as trees and walls, can obscure a signal beyond recognition. Typical SNR values are between 0 and 50, where 50 means an excellent signal. (SNR can be as high as 99, but I've never seen readings above 50 even in wide open sky.) In Figure 1, the green satellites indicate a strong signal, whereas the yellow satellite signifies a moderate signal. Satellite #1's signal is completely obscured. Listing Seven (available electronically) shows the interpreter after it is expanded to read satellite info.

A World-Class Interpreter

International readers may have spotted a subtle problem early on that was not handled in the source code I've presented—numbers were being reported in the numeric format used in the United States! Countries such as Belgium and Switzerland, which use different formats for numbers, require adjustments to the interpreter in order to work at all. Fortunately, the .NET Framework includes built-in support for converting numbers between different

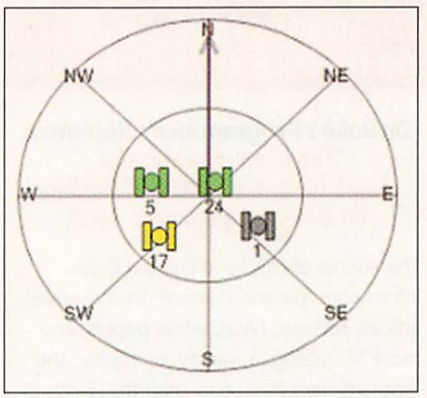


Figure 1: *Graphical representation of a $GPGSV sentence, where the center of the circle marks the current position and the edge of the circle marks the horizon.*

cultures, so the changes to the interpreter required are straightforward. In the interpreter, the only fractional values are speed and bearing, so two changes are necessary. The *NmeaCultureInfo* variable represents the culture used for numbers within NMEA sentences. The *Double.Parse* method is then used with this variable to convert speed into the machine's local culture. Listing Eight (available electronically) shows the completed interpreter, ready for use internationally.

Once you understand how an NMEA interpreter extracts words from sentences, you can harness the power of satellites to determine your location, synchronize your computer clock, find your direction, watch your speed, and point to a satellite in the sky on a cloudy day. The interpreter I present here also works with the .NET Compact Framework without any modifications. If sentences were also stored in a file, the interpreter can be used to play back an entire road trip. These are all great features, especially considering the small size of the class, but is this interpreter ready to drive your car and pilot an airplane? Not quite yet. There is one important topic remaining, which is required to make GPS applications safe for the real world—precision.

High Precision

Although the interpreter I've presented now works with handheld devices and supports international developers, it still isn't necessarily suitable for commercial use because it does not monitor precision. Without precision, applications could end up making unintelligent business decisions such as accidentally telling drivers to turn left into an alley, or worse. Clearly, the issue of "precision" must be addressed if GPS applications are to be smart enough for in-car navigation and reliable enough to autopilot an airplane.

ESSENTIAL INFORMATION FOR SERIOUS DEVELOPERS
On the Web at www.ddj.com

***Dr. Dobb's* Programmer's Resources**
The premier location for comprehensive technical information for every programmer's needs

The editors of *DDJ* have created these information-packed channels to showcase articles, reviews, FAQs, white papers, and more. Covering a variety of topics, the Programmer's Resources span the depth of software development from algorithms to real-time computing.

DATABASE DEVELOPMENT CHANNEL...

Few applications are more fundamental to business than databases. This channel keeps developers up to date on the latest in database technology, from embedded to distributed systems. The Database Development Programmer's Resource contains in-depth articles, white papers, FAQs, a discussion forum, related Web links and more!

www.ddj.com/topics/database/

DISTRIBUTED COMPUTING CHANNEL...

From Beowulf to CORBA, distributed computing is moving into the mainstream as more and more processing power is needed and the Net grows. This channel keeps developers up to date on the latest in distributed computing technology, including Real-Time CORBA and cluster computing.

www.ddj.com/topics/distrib/

ADDITIONAL CHANNELS...

Java / Linux / Algorithms / C/C++ Programming
Scientific Computing / Computer Security
Scripting and Alternative Languages
Patterns and Object-Oriented Design
Graphics Programming / Communications and Networking / Real-Time Computing
AI /Data Compression / UNIX / XML and Web Services / Python-Url / TLC-URL / Benchmarking and Software Testing / Basic / Perl

VISIT TODAY!
www.ddj.com

CMP
Dr. Dobb's JOURNAL

GPS devices calculate your position using a technique called "3D multilateration," which is the process of determining where several spheres intersect. In the case of GPS, each sphere has a satellite at its center; the radius of the sphere is the calculated distance from the satellite to the GPS device. Ideally, these spheres would intersect at exactly one point, causing there to be only one possible solution to the current location, but in reality, the intersection forms more of an oddly shaped area. The device could be located within any point in the area, forcing devices to choose from many possibilities. Figure 2 shows such an area created from three satellites (using the $GPGSV sentence). The current location could be any point within the gray-colored area. Precision is said to be "diluted" by this area, which leads to this article's second focus: dilution of precision. The monitoring and control of dilution of precision (DOP) is the key to writing high-precision applications.

DOP values are reported in three types of measurements: horizontal, vertical, and mean. Horizontal DOP (HDOP) measures DOP as it relates to latitude and longitude. Vertical DOP (VDOP) measures precision as it relates to altitude. Mean DOP, also known as Position DOP (PDOP), gives an overall rating of precision for latitude, longitude, and altitude. Each DOP value is reported as a number between 1 and 50, where 50 represents very poor precision and 1 means ideal accuracy. Table 1 lists what I believe to be an accurate breakdown of DOP values.

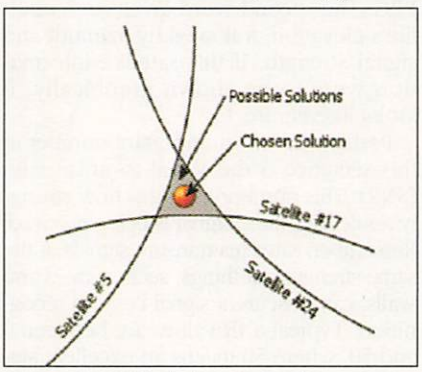


Figure 2: *GPS devices must choose one of several possible solutions to the current location.*

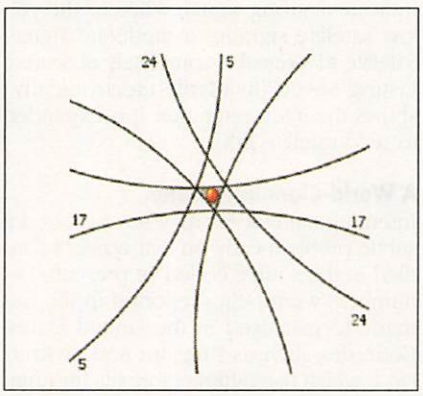


Figure 3: *Three more evenly distributed satellites are added to Figure 2, creating a high-precision environment where dilution of precision is low.*

Looking again at Figure 2, three satellites created a large area of possible solutions. This situation could be improved by two factors: adding more satellites to the fix, and using satellites evenly distributed throughout the sky. What would Figure 2 look like if the situation was improved like this? Figure 3 shows Figure 2 after three more evenly distributed satellites have been added. As a result, the DOP has been reduced.

Advanced GPS application developers could know if positional readings are precise enough to use just by looking at one $GPGSA sentence. Again, the best DOP ratings occur when there are several satellites involved in a fix and the satellites are evenly distributed throughout the sky and at separate elevations—hitting the GPS device from all angles, so to speak.

Determining Precision Needs

The next step is to determine the actual precision needs of an application. As a general rule, an HDOP value of 6 or less is recommended for any application that makes suggestions to the user based on the current location. For example, in-car navigation programs that tell the user to "turn left now" should ignore positional measurements when HDOP is greater than 6.

But is 6 really good enough? How can you figure out which HDOP values to use for your applications? To answer these kinds of questions, use this simple formula:

Accuracy of GPS Device * DOP =
Maximum Allowable Error

This formula uses *DOP* as a factor of error that, when combined with the accuracy of the GPS device being used, yields the maximum error allowed by a level of *DOP* in the form of a specific, measurable distance. Another general rule is that typical consumer GPS devices are capable of between 5–7 meters of accuracy without enhancements (such as DGPS or WAAS), or an average of 6 meters. Using the in-car navigation HDOP of 6 and a typical GPS device, the maximum error allowed is *6m * 6 = 36 meters*, or 118 feet. Given that a downtown city block is roughly 475 feet square, the maximum allowable error is about a quarter of a city block. This is precise enough to make sure that drivers turn at the correct road. On the other hand, an HDOP of 12 results in an allowable error of half a city block (237 feet), which could cause drivers to turn down an alley accidentally. So, using the

formula, it is possible to use an HDOP greater than 6 for in-car navigation, but not by much.

The trick to using this formula is researching real-world distances, especially the smallest important distances. To demonstrate, take a look at golf. Does golf require more precision than in-car navigation? A golf program needs to tell users which golf club to use to make the best shot. Some research into important golf distances finds that for most players, there is a regular distance interval between clubs of about 10–15 yards. Therefore, a golf program needs no more than 10 yards (9.1 meters) of allowable error to consistently suggest the right club. When 9.1 meters is put into the formula as *Maximum Allowable Error*, the maximum HDOP comes out to 3. So, golfing applications do indeed require about twice the precision as in-car navigation systems.

Why not skip the formulas and always enforce an HDOP of 1? This looks like a reasonable practice, but greater precision requires greater satellite visibility. An in-car navigation system will probably not get an HDOP of 1 (or even 3) downtown because signals are being obscured by buildings. If the enforced HDOP is too small, the application throws out too many positional readings and just sits there while drivers lose patience. Golfing applications, on the other hand, can realistically enforce a small HDOP because they operate in more open sky. The golfer's PDA is likely to have enough sky to pick up several evenly distributed satellites, unless their ball is in the middle of the woods, in which case they're on their own.

To summarize, successful GPS software developers use the formula to determine the largest possible DOP number. This ensures that the application minimizes the most problems due to inaccuracy while at the same time allowing the application to function in the poorest possible satellite visibility conditions. This practice maximizes the value and versatility of any GPS application.

Enforcing Precision

Once the precision needs of a GPS application have been determined, it's time to find out what source code is necessary to extract and enforce maximum DOP values. All DOP measurements are packaged into the $GPGSA sentence every few seconds. Here is a sample of a $GPGSA sentence:

```
$GPGSA,A,3,11,29,07,08,5,17,24,,,,,,2.3,1.2,2.0*30
```

The last three words of this sentence are 2.3, 1.2, and 2.0, representing mean, horizontal, and vertical DOP, respectively. This sentence represents a high-precision environment (suitable enough for both golfing and driving). Using Listing Eight (available electronically), a method called *ParseGPGSA* is added, which extracts DOP values and reports them via three events—*HDOPReceived*, *VDOPReceived*, and *PDOPReceived* (see Listing Nine, available electronically).

Enforcing DOP values is the easiest part of the programming process because enforcing precision is a matter of ignoring positional measurements until HDOP is at (or below) the maximum. This can be done in a single *if* statement. To demonstrate this, I've written a small app (Listing Ten, available electronically) that uses the NMEA interpreter to enforce high precision.

Conclusion

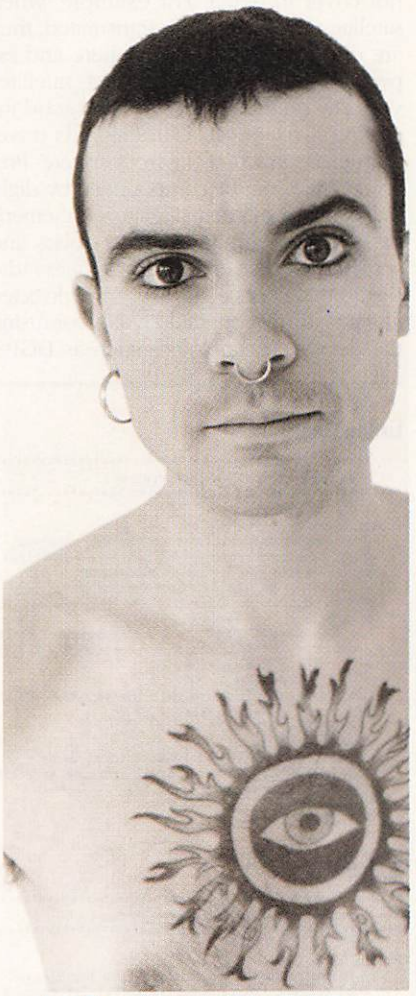
Controlling dilution of precision is the key to writing commercial-grade GPS applications. A basic formula can be applied to determine the maximum allowable error for a particular application. The maximum allowable error should be the greatest possible value that minimizes accuracy problems while maximizing operational conditions.

While this article covered programming techniques for controlling precision, there

DOP	Rating	Description
1	Ideal	This is the highest possible confidence level to be used for applications demanding the highest possible precision at all times.
2–3	Excellent	At this confidence level, positional measurements are considered accurate enough to meet all but the most sensitive applications.
4–6	Good	Represents a level that marks the minimum appropriate for making business decisions. Positional measurements could be used to make reliable in-route navigation suggestions to the user.
7–8	Moderate	Positional measurements could be used for calculations, but the fix quality could still be improved. A more open view of the sky is recommended.
9–20	Fair	Represents a low confidence level. Positional measurements should be discarded or used only to indicate a very rough estimate of the current location.
21–50	Poor	At this level, measurements are inaccurate by as much as half a football field and should be discarded.

Table 1: *Jon's interpretation of dilution of precision values.*

Went to find himself.



Left you with 300K lines of C++ using macros to look like Pascal and a Linux box you can't boot.

www.scitools.com

Tools that help you understand and maintain impossibly large bodies of source code.

Supported Languages:
* *Ada 83 and Ada 95*
* *Java*
* *ANSI C, K&R C, and C++*
* *FORTRAN 77, 90, 95*
* *JOVIAL*
* *Pascal*
* *Assembly*
* *Can create custom languages on request*

are several other technologies that I did not cover in detail. For example, when satellite radio signals are transmitted, they are distorted by the troposphere and especially the ionosphere. In fact, satellites very low on the horizon are not good for getting a fix because the signals travel through so much of the troposphere. Precision errors can be compounded by slight inaccuracies in each satellite's ephemeris (its actual position in orbit), its clocks, and something called "multipath" where radio signals that bounce off objects are detected along with the original signal, confusing the device. Technologies such as DGPS, Internet DGPS, SisNet, WAAS, and EGNOS attempt to minimize most of these precision errors by providing corrective data that, when integrated with a device's own data, can improve accuracy down to as little as 1 meter in good conditions. A lot of consumer devices are now being shipped with built-in support for WAAS, but WAAS is still considered experimental and frequently broadcasts that it is in test mode.

There are some applications that require so much precision that they escape the scope of this article and enter the realm of expensive dual-channel receivers and custom ground stations. Yet, GPS is successfully landing planes while you read this, and it is only a matter of time before precision down to a centimeter becomes cost-effective for industry. When that happens, I believe it will cause a critical mass that opens the door to some truly amazing and helpful things— automated construction machines, coupons on your cell phone for that restaurant across the street, traffic-control systems that actually take the wheel...and self-guiding golf balls.

DDJ

Listing One

```
'*************************************************************
'**  The core of an NMEA interpreter
'*************************************************************

Public Class NmeaInterpreter
    ' Processes information from the GPS receiver
    Public Function Parse(ByVal sentence As String) As Boolean
        ' Divide the sentence into words
        Dim Words() As String = GetWords(sentence)
        ' Look at the first word to decide where to go next
        Select Case Words(0)
            Case "$GPRMC"     ' A "Recommended Minimum" sentence was found!
                ' Indicate that the sentence was recognized
                Return True
            Case Else
                ' Indicate that the sentence was not recognized
                Return False
        End Select
    End Function
    ' Divides a sentence into individual words
    Public Function GetWords(ByVal sentence As String) As String()
        Return sentence.Split(",")
    End Function
End Class
```

Listing Two

```
'*************************************************************
'**  Extracting information from a sentence
'*************************************************************

Public Class NmeaInterpreter
    ' Raised when the current location has changed
    Public Event PositionReceived(ByVal latitude As String,
                                  ByVal longitude As String)
    ' Processes information from the GPS receiver
    Public Function Parse(ByVal sentence As String) As Boolean
        ' Look at the first word to decide where to go next
        Select Case GetWords(sentence)(0)
            Case "$GPRMC"    ' A "Recommended Minimum" sentence was found!
                Return ParseGPRMC(sentence)
            Case Else
                ' Indicate that the sentence was not recognized
                Return False
        End Select
    End Function
    ' Divides a sentence into individual words
    Public Function GetWords(ByVal sentence As String) As String()
        Return sentence.Split(",")
    End Function
    ' Interprets a $GPRMC message
    Public Function ParseGPRMC(ByVal sentence As String) As Boolean
        ' Divide the sentence into words
        Dim Words() As String = GetWords(sentence)
        ' Do we have enough values to describe our location?
        If Words(3) <> "" And Words(4) <> ""
                            And Words(5) <> "" And Words(6) <> "" Then
            ' Yes. Extract latitude and longitude
            Dim Latitude As String = Words(3).Substring(0, 2) & "°"
                                                    ' Append hours
            Latitude = Latitude & Words(3).Substring(2) & "'"
                                                    ' Append minutes
            Latitude = Latitude & Words(4)          ' Append the hemisphere
            Dim Longitude As String = Words(5).Substring(0, 3) & "°"
                                                    ' Append hours
            Longitude = Longitude & Words(5).Substring(3) & "'"
                                                    ' Append minutes
            Longitude = Longitude & Words(6)        ' Append the hemisphere
            ' Notify the calling application of the change
            RaiseEvent PositionReceived(Latitude, Longitude)
        End If
        ' Indicate that the sentence was recognized
        Return True
    End Function
End Class
```

Listing Three

```
'*************************************************************
'**  Detecting and handling NMEA errors
'*************************************************************

Public Class NmeaInterpreter
    ' Raised when the current location has changed
    Public Event PositionReceived(ByVal latitude As String,
                                  ByVal longitude As String)
    ' Processes information from the GPS receiver
    Public Function Parse(ByVal sentence As String) As Boolean
        ' Discard sentence if its checksum does not match calculated checksum
        If Not IsValid(sentence) Then Return False
        ' Look at the first word to decide where to go next
        Select Case GetWords(sentence)(0)
            Case "$GPRMC"     ' A "Recommended Minimum" sentence was found!
                Return ParseGPRMC(sentence)
            Case Else
                ' Indicate that the sentence was not recognized
                Return False
        End Select
    End Function
    ' Divides a sentence into individual words
    Public Function GetWords(ByVal sentence As String) As String()
        Return sentence.Split(",")
    End Function
    ' Interprets a $GPRMC message
    Public Function ParseGPRMC(ByVal sentence As String) As Boolean
        ' Divide the sentence into words
        Dim Words() As String = GetWords(sentence)
        ' Do we have enough values to describe our location?
        If Words(3) <> "" And Words(4) <> "" And Words(5)
                                    <> "" And Words(6) <> "" Then
            ' Yes. Extract latitude and longitude
            Dim Latitude As String = Words(3).Substring(0, 2) & "°"
                                                    ' Append hours
            Latitude = Latitude & Words(3).Substring(2) & "'"
                                                    ' Append minutes
            Latitude = Latitude & Words(4)          ' Append the hemisphere
            Dim Longitude As String = Words(5).Substring(0, 3) & "°"
                                                    ' Append hours
            Longitude = Longitude & Words(5).Substring(3) & "'"
                                                    ' Append minutes
            Longitude = Longitude & Words(6)        ' Append the hemisphere
            ' Notify the calling application of the change
            RaiseEvent PositionReceived(Latitude, Longitude)
        End If
        ' Indicate that the sentence was recognized
        Return True
    End Function
    ' Returns True if a sentence's checksum matches the calculated checksum
    Public Function IsValid(ByVal sentence As String) As Boolean
        ' Compare the characters after the asterisk to the calculation
        Return sentence.Substring(sentence.IndexOf("*") + 1) =
                                                GetChecksum(sentence)
    End Function
    ' Calculates the checksum for a sentence
    Public Function GetChecksum(ByVal sentence As String) As String
        ' Loop through all chars to get a checksum
        Dim Character As Char
        Dim Checksum As Integer
        For Each Character In sentence
            Select Case Character
                Case "$"c
                    ' Ignore the dollar sign
                Case "*"c
                    ' Stop processing before the asterisk
                    Exit For
                Case Else
                    ' Is this the first value for the checksum?
                    If Checksum = 0 Then
                        ' Yes. Set the checksum to the value
                        Checksum = Convert.ToByte(Character)
                    Else
                        ' No. XOR the checksum with this character's value
                        Checksum = Checksum Xor Convert.ToByte(Character)
                    End If
            End Select
        Next
        ' Return the checksum formatted as a two-character hexadecimal
        Return Checksum.ToString("X2")
    End Function
End Class
```

DDJ

MidiRecorder: An Exercise in Code Reuse

When reusable code becomes a reality

AL STEVENS

Those of you who hang out at piano bars know the routine. The piano player passes the microphone around and the patrons take turns belting out tunes. Most *DDJ* readers probably don't do that. At least, I hope you don't.

The typical piano bar patron is from an earlier generation, even earlier than mine. I'm a retired programmer and writer who sits at the business end of a piano bar and I'm usually the youngest dude in the joint. The patrons all think they're Streisand or Sinatra and my job is to encourage their fantasies—keep them drinking and keep the tip jar growing. I privately refer to myself as "organic karaoke." It's how a serious piano player keeps working between jazz gigs. If you want to hear about that life, go to my web site (http://www.alstevens.com/), click on "Tunes" and listen to a tune I wrote called "Open Mic."

But that's not the point of this article.

This article is about code reuse and how I used it to write a program that actually makes the contents of the tip jar get bigger. I can hear you now. "Huh?"

Code Reuse

Taking off the piano player hat and donning the programmer hat, I offer the following opinion: For years, code reuse was the urban legend of programming. If you designed proper classes and algorithms, other people would use them, too—or so the story went. Build it and they will come. Only we didn't and neither did they.

Al is DDJ's senior contributing editor and a professional musician. He can be contacted at al@alstevens.com.

Reusable code is not a new idea. It's an old one that has yet to be fully realized.

In my years as a programming consultant, I rarely saw reusable code being written in the workplace. Programmers in the trenches are too busy meeting unrealistic schedules to worry about writing reusable code. You know who does worry about it, however? Writers, that's who.

Those of us who feed on the reusable code mantra earn our keep by publishing code that is actually nearly reusable. It takes extra effort to design code that works outside its original domain and for its original purpose. Code that others can use is the programmer's version of a tip jar. Write code for other programmers and you might even get paid for it.

Back to the Piano Bar

Those singer-wannabes often tell the piano player, "I wish I could take you home," as if every piano player's wildest dream is to spend our declining years in the patronage of some diva who wakes up singing "Summertime, and the livin' is easy..." In my nightmares. But suppose you can make a more appealing offer. Don't take me home; take my accompaniments home. That's what she really wants, anyway. You hope.

Each amateur singer has a small repertoire of tunes and a savvy pianist learns who sings what and in what key. With some extra effort, you could go into a studio and record accompaniments for them all. But you don't want to put in that extra effort. Why spend hours in a studio laying down tracks just to get barflies to toss an extra buck in the jar? Why not record the accompaniment in real time while you play it in the piano bar?

Laying Down Tracks in Real Time

Obviously, an audio recording wouldn't do. That would capture all the crowd noise and the actual vocal along with the piano accompaniment. The last thing you want is for patrons to realize in the cold light of a sober morning how bad they sound. Your following would disappear. Even if they sing well—some actually do—they want accompaniments with which to practice, not recordings of themselves.

A laptop and MIDI provide the answer. A pianist can record a MIDI sequence if the piano supports the MIDI protocol. Any sequencer program can handle it. But there's a catch. Sequencer programs—at least their recording functions—are meant to be used in the studio. They require a lot of attention and user interaction, which a performing entertainer cannot provide while at the same time dealing with patrons. This problem calls for a program that works on its own. And such is MidiRecorder, the program I came up with.

> "Programmers in the trenches are too busy meeting unrealistic schedules to worry about writing reusable code"

MidiRecorder

MidiRecorder runs in a PC laptop with its MIDI IN port attached to the MIDI OUT port of an electronic piano keyboard. MidiRecorder just sits there waiting for you to play something. When you do, it starts collecting MIDI event data. When MidiRecorder sees that you have not played something for a specified number of seconds, it writes the collected event data to a Standard MIDI Format (SMF) file and

stands by waiting for you to start playing again. The idea is that, after a certain period of virtual silence, the tune must be over. You don't have to touch the laptop or even have it in sight. You can keep your eye on the tip jar and schmooze with the patrons, which is what you ought to be doing instead of operating a computer anyway.

MidiRecorder does have a very simple user interaction that lets you change destinations for the SMF files with two keystrokes. That way, you can keep the accompaniments for each singer separate.

When you go on break, you write the collected files onto diskettes and give them to the customers. Or sell them. With an autographed picture. And a CD of your greatest hits.

The source code for MidiRecorder is available electronically; see "Resource Center," page 5. An executable version and HTML user's guide is at http://www.alstevens.com/midifitz/.

You might ask, "What is the average Joe or Jane going to do with SMF files?" Guess what? All those over-the-hill crooners not only think they're Sinatra, they think they're Pournelle, too. They all have state-of-the-art desktop computers and, if you encourage them, will regale you with their extensive knowledge of XP, eBay, Yahoo, RAM, ROM, DVD, MP3, and all that. They can handle it. Just give them their diskettes and tell them to double-click the icons to hear their accompaniments played back through their super-whammy SoundBlasters. It's up to them to turn up their hearing aids and remember which tunes they sang in what order or to at least recognize the tunes from the accompaniments. Then they can entertain their spouses, grandchildren, and pets to their hearts' content. "Feelings, nothing more than feelings..."

But, then you ask, "If you do not want to devote studio time to the benefit of your aged groupies, why are you willing to labor for days and hours hacking out a program?" Aha. Such is the beauty of reusable code. The program I described took about an afternoon to write. How so? Because I used a lot of code I had written in the past. More specifically, I used a lot of code I had published in the past. That's the secret. Write publishable code—even if you never intend to publish it.

MidiRecorder Requirements

MidiRecorder needs to do the following:

1. It must capture MIDI IN events from the piano keyboard.
2. It must pass those events through to a selected MIDI output device.
3. It must save those events in an internal queue that includes a time stamp.
4. It must, upon a timeout, write the queued events to an SMF file.

All those requirements have been met in other programs I wrote and published over the years in the "C Programming" column in *DDJ*.

- A project named MidiFitz read the MIDI IN port to parse keyboard chord data and produce realtime rhythm section accompaniment

"Publishing code is like programming with no pants on"

- A class library named midifile manages SMF file input/output.
- A project named Jukebox is a sequencer that includes the algorithms for managing MIDI event data timing and writing to a MIDI OUT port.

Because I wrote that earlier code to be read by programmers besides being compiled by compilers and run by users, I wrote it better.

Reusing Unusable Code

We've all written a lot of code that doesn't look good or work well. When I hammer out a program to use here, I don't worry much about it having bugs that I can work around. The program is not for distribution. It's for me. I know its idiosyncrasies and can live with them. Such code is unusable outside the program for which it is written. Fixing it up and making it pretty is more trouble than it is worth. That is, unless the program turns out to be useful for other people, in which case I might go back in, clean it up and distribute it.

Back when C++ was making its initial inroads among mainstream programming, I sat in the lobby at a computer convention with P.J. Plauger discussing the notion of the object-oriented programming paradigm. P.J. said something that stuck with me. "I'm a procedural kind of guy," referring to the way he liked to write code at the time. As it turns out, so am I. Most of the programs I write just for me are a hybrid of procedural code with only enough of an object-oriented slant to accommodate any existing classes I want to use. That's no surprise when you consider that I programmed for 30 years without knowing what an object was.

But when it's time to overhaul a program for public consumption, I do what I ought to have done in the first place; I implode all that raw, exposed procedural code into an orderly, encapsulated class architecture. In virtually every case, I improve the program, sometimes finding bugs I didn't know were there and always making the program more reliable, extensible, and maintainable. And readable. Which is the same thing.

Publish Properly or Perish Pantsless

I've said this before. Publishing code is like programming with no pants on. Everyone gets to see what you have. With programmers worldwide poring over your every line, the code not only has to work, it better look good. And it better obey conventional rules of correct coding. In other words, it ought to have all those lofty and noble attributes that production code rarely has.

Otherwise, you get a lot of e-mail that points out your transgressions.

There used to be (and still are) a lot of formal "methodologies" you could buy into to address the problems of busted schedules and buggy releases. Most such methodologies are little more than vehicles with which their designers get fat consulting and seminar fees. When the methodologies fail to make things better, their purveyors simply blame the users; they must not have applied the methodology properly. It is my observation that when deadlines loom and management puts the pressure on, programmers throw out the disciplines and just hunker down and code furiously day and night.

Of them all, one particular philosophy stands the test of time. In 1971, Gerald Weinberg published *The Psychology of Computer Programming* (Dorset House Publishing, 1998; ISBN 0932633420) in which he describes a system of peer code review wherein code projects are regularly reviewed by panels of each programmers' peers in search of bugs, questionable idioms, and other problematic programming practices. He calls it "egoless programming" because you have to check your ego at the door during your turn in the code review barrel.

Publishing code in books and magazines opens the door for lots of source code peer review. All your readers get to

review your code. Tens of thousands of them. They paid for it, they expect it to help them do their jobs, and they delight in finding something wrong with it. If your ego is fragile, don't publish code.

(Don't involve yourself in an open-source project, either, if your ego is easily bruised. That environment provides what is probably the ultimate source code peer review. But that's another subject altogether.)

Authors, at least good ones, learn to pore over their own code anticipating what all those critical reader/reviewers might find. And they do that even after the code works. If it ain't broke, don't fix it? Well, if it ain't pretty, paint it.

Once you've published a program, your mistakes belong to posterity and haunt you forever. "If only I'd have seen that before it went to press..."

That's the secret; write code as if the world will not only use it but read it, too. Write it for those programmers who like to catch a published author with his pants down.

Component Code?

Component code encapsulates abstractions into packages with a common user interface—the users in this case being the programmers who use the components in their programs. There are many specifications for how to build a component. Some component specifications are language independent; some are platform independent. The concept of components could have been implemented all along in existing languages, and often was, but it took the Internet to provide a mass medium for component sharing and distribution.

I'm not talking about that kind of reusable code, however. I believe that component development and use foreshadows how all programming will be done some day. But not just yet.

My definition of reusable code does not hide details from the programmer. It does not involve black boxes or boxes of any other hue. It comprises lines of code that worked in the past, that you can cut out of a working program and paste into another program, and that you trust and is neither difficult to use nor to port.

When functionality is nicely separated from the rest of the program in encapsulated classes, that's all the compartmentalization of components you need to regard the function's code as reusable.

Reusable == New and Improved

Almost every time I reuse one of my own classes, I see more ways that it can become more generic and less specific to problem domains it served in the past. So, I upgrade the class to make it even more reusable for future projects.

Which is why if you compare the code in MidiRecorder to that in published versions of previous projects, you see some differences. Consider them improvements.

If I'm feeling particularly conscientious, I might retrofit those upgraded classes into my copies of old projects that used them. That exercise seldom makes the old programs work better, but it validates the new, improved classes' reusability.

MidiRecorder Internals

Back to the program at hand. You'll have to get the source code to follow along. It compiles with GCC, and I've included a project file for the Quincy 2002 IDE (http://www.alstevens.com/quincy2002/). Earlier versions compiled with Visual C++, too. I don't know whether the latest version does because I've made no effort to maintain compiler portability. But it's a command line program that uses the Win32 API for MIDI functions and Standard C++ for file and console input/output, so it ought to work okay.

The main.cpp source code file contains the main function along with some other housekeeping functions. A close examination of main.cpp reveals the procedural programmer in me. Can't shake it. MidiRecorder uses command-line arguments to set parameter values. I did that so I could launch it by clicking a batch file while sitting at the piano. I like to keep the laptop out of sight from the patrons. Otherwise, they'll want to tell me all about their hard drives or something.

The main function gets the command-line arguments and, if everything isn't specified on the command line, calls functions that prompt users to specify input and output MIDI devices. Then it instantiates an object of type *midiRecord* and calls the *midiRecord::record* member function to get things rolling. That's the last thing the main function does. The program now runs until the piano player, er, user terminates it with a Ctrl-C.

The *midiRecord* class includes data members of types *MidiIn* and *MidiOut*. These classes are from previous projects and they needed little or no modification to be used by MidiRecorder. They also represent some of the most complex parts of the program, the stuff you might spend weeks and months researching, developing, and debugging if you didn't already have classes that do what the program needs. They handle MIDI event data from, in this case, the piano keyboard and MIDI event data to whatever MIDI output device to which you want MidiRecorder to pass input events. I sometimes use the output feature as a THRU channel to send keyboard data to a sound module whenever I don't like the acoustic piano sounds produced by the keyboard itself.

The *midiRecord* class also has a data member pointer to type *SMFout*. This class, if anything, supports a more complex operation than either *MidiIn* or *MidiOut*. It takes care of writing an SMF file. It derives from the base class *MIDIFile*, which I developed years ago to support SMF file input/output, and which I published in the "C Programming" column. I love this class. I use it often and it always saves me a lot of time. An application derives a class from MIDIFile with the application class's constructor calling the MIDIFile constructor with either a *std::ifstream* or *std::ofstream* argument. Which one you use determines whether the derived class is for SMF file input or output. In this case the *SMFout* class is for output. The derived class overrides some *MIDIFile* member functions and calls other *MIDIFile* member functions. Which functions it overrides and which ones it calls depends on whether the derived class is for input or output. One base class handles either operation. Comment in midifile.h tells the programmer which member functions to override or call.

The *midiRecord::record* function is what gets things going. It uses the Win32 *CreateEvent* function to build a semaphore with which to control its polling loop. Then it sets a timer and goes into a loop polling for things to happen.

The *midiRecord::Poll* function watches for input from the computer keyboard and for events triggered by timeouts and MIDI input events. A close reading of the source code should reveal how all this works. It isn't my purpose here to teach you all the nitty-gritties of MIDI sequencing, only to make the point that, by leveraging code that I develop with such levers in mind, I was able to build a complex program in a very short time. In fact, the classes in this program have much of what a programmer needs to build a MIDI-only sequencer program along the lines of Cakewalk but without the audio support. Just add a user interface and off you go.

Conclusion

Readers should be no strangers to reusable code. Your application framework of choice is just one example. (Mine are *std::cin* and *std::cout*, so don't follow my example there.) So are the Standard C and C++ libraries. The lesson here is that, by writing your own code to be reusable, you can help yourself down the line when that next project needs something you've already built.

DDJ

Perl, VMWare, & Virtual Solutions

Building virtual build and test farms

SAM TREGAR

Krang is an open-source content-management system that runs on a variety of operating systems, including various Linux and BSD flavors. To make Krang easy to install, we produce binary distributions for each supported platform. For example, if you're running Redhat Linux 9 with Perl 5.8.4, you would download this file to install Krang:

krang-1.020-Redhat9-perl5.8.4-i686-linux.tar.gz

A binary distribution contains the Krang code, written in Perl, an Apache/mod_perl web server, and all the external Perl modules needed to run Krang. All you need to supply is a base operating-system installation and MySQL 4.0.13 or later. If there are other requirements on your platform, they're listed in a platform-specific README file.

As we ported Krang to more and more platforms, generating these binary distributions for each release became a real chore. Each build would have to be performed by hand on each target platform. Worse, each distribution needs to be tested to make sure Krang still works on all our platforms.

My solution is the Krang Farm (http://krang.sf.net/), an automated build and test system created using VMWare GSX Server and Perl. With the Krang Farm, I can enter a single command to build and test Krang across all our supported platforms, all on a single physical machine. In this article, I'll explain how the farm works and how you can create your own virtual farms using VMWare GSX Server and Perl.

VMWare Backgrounder

VMWare (http://vmware.com/) produces virtual-machine software. A virtual machine is basically a simulated machine that runs its own operating system. Each virtual machine has its own virtual hardware (processor, hard drive, video card, and so on), and is completely separate from the host system. The real machine that runs the VMWare software is known as the "host," while the virtual machines are "guests." Thus, the term "guest operating system" refers to the operating system running inside a virtual machine.

VMWare's software isn't an emulator like Bochs or Virtual PC. Instead, VMWare uses the built-in virtualization hardware in all x86 processors. This lets the virtual machines run nearly as fast as software that runs directly on the real hardware. However, it does have one major drawback—the virtual machines are all x86 systems. VMWare won't run PowerPC or Sparc virtual machines, for example. (See the sidebar "Why Not Build a Real Farm?")

The first VMWare product I used was VMWare Workstation, which I use to run Microsoft Windows on my Linux desktop. When I need to test a web application under IE or open an Excel spreadsheet that won't work with OpenOffice, VMWare Workstation is indispensable. A coworker uses VMWare Workstation in the reverse situation—to do development on a Linux guest running VMWare on a Windows host.

> "VMWare offers two APIs—VMPerl for Perl scripting on Linux and Windows and VMCOM for COM programming on Windows"

I've also used VMWare Workstation to do some software development for the Bricolage project. I set up virtual machines running a few guest operating systems supported by Bricolage: FreeBSD, Debian Linux, and Redhat Linux. I used these

Sam is a Perl programmer at the PIRT Group (http://thepirtgroup.com/) and author of Writing Perl Modules for CPAN *(Apress, 2002). He can be contacted at sam@tregar.com.*

EARLY BIRD
Register by
February 18th
and save up to
$300

SUPER
EARLY BIRD
Register by
January 14th
and save up to
$500

OVER 200 SESSIONS IN

- .NET
- C++
- Java
- Modeling & Design
- People, Process & Methods
- Requirements & Analysis
- Scripting
- Security
- System Security
- Testing & Quality
- Web Services
- XML

Over 164 NEW sessions

PLUS

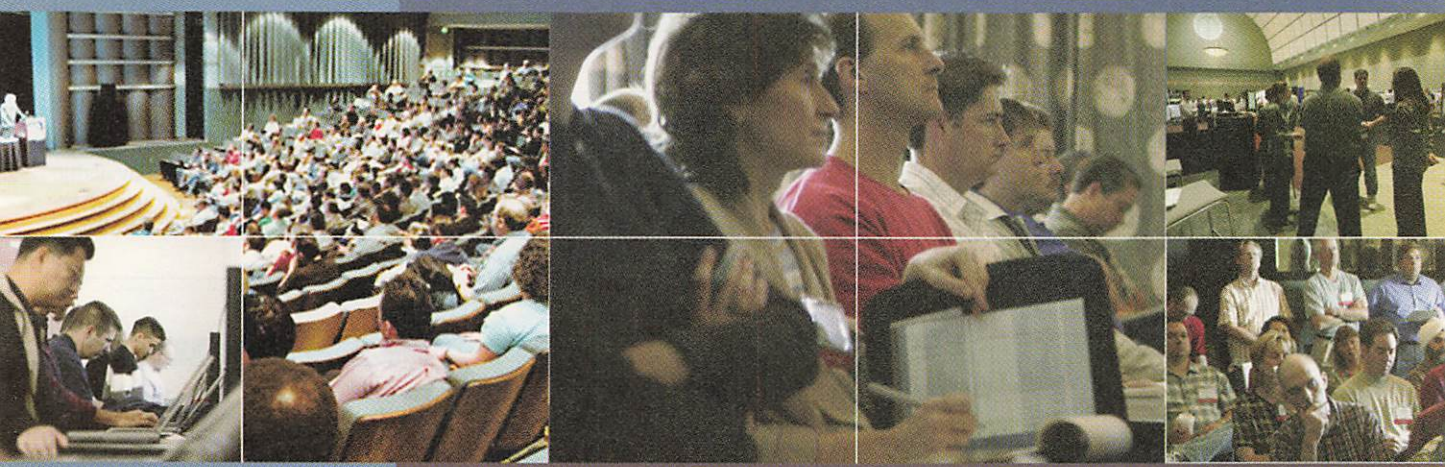
Expo, Keynotes, Panels,
Case Studies, BOFs,
and NEW special events

Special events include:

DDJ EXCELLENCE IN
PROGRAMMING PANEL

Panelists include:

- Don Chamberlin
- James Gosling
- Guido van Rossum
- Alexander Stepanov
- and other past recipients



SD WEST 2005

DEVELOP AN ADVANTAGE

March 14-18, 2005
Santa Clara, CA

Register today at www.sdexpo.com

(continued from page 60)
machines to test the Bricolage installation system while it was under development.

However, for this project, VMWare Workstation just won't work. First, it requires a local X server and won't work over a network connection. Second, it doesn't offer a scripting interface to automate operations on the virtual machines.

VMWare's entry-level server product, VMWare GSX Server, remedies both these deficiencies. It works on a headless server and provides a complete scripting system via the VMPerl and VMCOM APIs. This is the product I used to produce the Krang Farm. The rest of the article will refer to VMWare GSX Server simply as "VMWare."

```
#!/usr/bin/perl -w
use VMware::VmPerl::Server;
use VMware::VmPerl::ConnectParams;

# connect to the server using all the default settings
$server = VMware::VmPerl::Server::new();
$server->connect(VMware::VmPerl::ConnectParams::new()) or
  die "Could not connect to server: ", ($server->get_last_error())[1];

# get a list of virtual machines
@vm_list = $server->registered_vm_names();
die "Could not get list of VMs from server: ", ($server-
    >get_last_error())[1]
  unless @vm_list;

# print them out
print "$_\n" for @vm_list;
```

Example 1: *enumerate_vms.pl lists all virtual machines.*

```
$ perl enumerate_vms.pl
/var/lib/vmware/Virtual Machines/Redhat7_3-0/Redhat7_3.vmx
/var/lib/vmware/Virtual Machines/Redhat7_3_i686/Redhat7_3_i686.vmx
/var/lib/vmware/Virtual Machines/Redhat9/Redhat9.vmx
/var/lib/vmware/Virtual Machines/Redhat9_i686/Redhat9_i686.vmx
/var/lib/vmware/Virtual Machines/Fedora1/Fedora1.vmx
/var/lib/vmware/Virtual Machines/Fedora2/Fedora2.vmx
```

Example 2: *Output from enumerate_vms.pl.*

Why Not Build a Real Farm?

The biggest reason to build a virtual farm rather than a real farm is cost. At my company, we can host and administrate a single machine for approximately $2400 per year. The cheapest new machine that meets our hosting requirements costs at least $3000. Thus, to build a farm of six machines costs $18,000 up-front and $14,400 per year. Each new machine will cost another $3000 plus $2400 per year.

Contrast this with the virtual farm. The machine running the Krang farm costs $4275; it's a ProLiant DL360 G3 from Hewlett-Packard with dual 2.8-GHz Intel Xeon processors and 1.5 GB of RAM. The VMWare GSX Server software costs $2500. It costs $2400 per year to host and administrate, just like any other machine on our network. Thus, the total cost to set up the virtual farm is $6775 and $2400 per year. That's already much lower than setting up six machines in a real farm. But even better, adding a new machine to the virtual farm is free. It doesn't require any new hardware or administration overhead.

Another reason to build a virtual farm is the added flexibility. Adding new machines to a real farm takes time; a new machine must be ordered and set up on the network. Adding a new machine to the virtual farm can be done in just a few hours, depending only on how long the operating-system installation takes to run.

However, a real farm has advantages over a virtual farm. First, it's likely to be faster. Because all the machines in the farm can run independently, it will scale better as machines are added. Since runtime performance isn't very important for a build and test farm, this wasn't an issue for the Krang Farm project. Second, a single machine means a single point of failure. It's much more important to have a working backup system when all your eggs are in one basket!

—S.T.

VMWare can be controlled through two interfaces—the web-based VMWare Management Interface and the Virtual Machine Console rich client. Figure 1 shows the VMWare Management Interface running in the FireFox web browser. This interface lets you provision new machines, reconfigure existing ones, start machines, and stop machines. It also has links to download the Virtual Machine Console, which runs as a native GUI on Linux and Windows.

The Virtual Machine Console is the interface used to set up a new machine because it gives you access to the virtual console. Users of VMWare Workstation will be at home in the interface as it is virtually identical. The only significant difference is the need to log in to the server before accessing the guest systems. Figure 2 shows the Virtual Machine Console accessing a Redhat Linux 9 guest at the beginning of the installation process.

Scripting VMWare

VMWare offers two APIs—VMPerl for Perl scripting on Linux and Windows, and VMCOM for COM programming on Windows only. Since I'm primarily a Perl programmer, I went straight to the VMPerl API.

Example 1 shows a simple script called "enumerate_vms.pl" that lists all available virtual machines. Example 2 shows the result of running this script on my VMWare server.

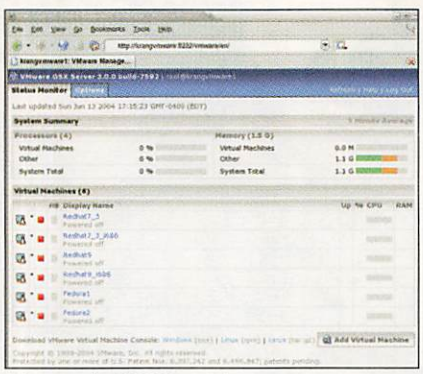


Figure 1*: VMWare Management Interface running in the FireFox browser.*

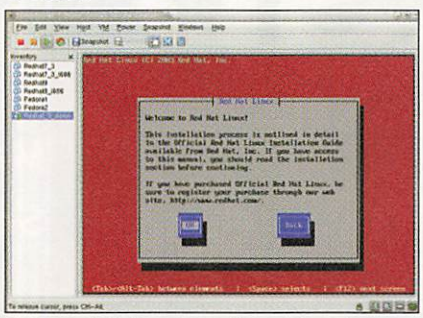


Figure 2: *Virtual Machine Console accessing a machine installing Redhat Linux 9.*

As you can see, VMPerl is an object-oriented API. Example 1 works by creating a *VMware::VmPerl::Server* object and calling *connect()* on it. The *connect()* method takes a *VMware::VmPerl::ConnectParams* object, which I created using the *new()* method. Not passing any arguments to *new()* causes *VMware::VmPerl::ConnectParams* to use all the default values for hostname, port, username, and password. If this script were run on a different machine from the VMWare installation, then these values would have to be filled in. Finally, a call is made to the *registered_vm_names()* method on the server object. This returns a list of virtual machine identifiers, which are in fact the paths of configuration files.

The VMPerl API contains methods for starting and stopping machines, as well as a mechanism for exchanging data with programs running inside the guest operating system. Throughout, I found the API to be well designed and easy to pick up. The excellent documentation and plentiful examples that come with the software are a great help.

Time Travel

Software testing becomes much harder if the results of one test run can effect the results of the next run. It is very hard to ensure this won't happen because bugs have a way of defying your specifications—very hard, that is, unless you can travel back in time. Imagine setting up your system in a "known-good state." Then, every time you want to run a test, you just travel back in time and run your test on your system in that state.

VMWare lets you do just that. VMWare's virtual storage system has a "nonpersistent" mode. This means that every time the virtual machine is booted, its hard drive has the contents it had when it went into nonpersistent mode, the contents of the known-good state. During the run, changes are made on disk, but nothing lasts beyond the next boot. This gives each test run a clean slate. No matter how badly it bombs, there's no way it can affect the next run.

Running Commands On Virtual Machines

So far, I've described how new machines are created and how they can be controlled programmatically via the VMPerl API. The final piece needed to run automated tests and builds on virtual machines is a way to run commands and get output from software running on the machines.

The simplest way to solve this problem is to use a network connection to control a shell session running on the virtual machine. Telnet and RSH would have worked fine, but I used SSH because most modern operating systems have SSH running after installation.

I used the Expect module to interact with shell sessions running inside the guest operating systems (http://www.cpan.org/authors/id/R/RG/RGIERSIG/). Expect is a Perl implementation of the venerable Expect TCL system created by Don Libes and it operates much the same way. The Perl implementation was written by Austin Schutz and Roland Giersig. You provide Expect a command to spawn and it handles setting up a pseudoterminal (pty) for the new process. Then you can run pattern matches on the output of the command and send the command input.

Example 3 shows via ssh how to run the *date* command on another machine (it could be virtual or real). When prompted, the script provides the user's password. The script then captures the output from

```
#!/usr/bin/perl -w
use Expect;

# connection parameters
$SERVER = 'Redhat9';
$USER   = 'krang';
$PASS   = 'krang';

# spawn the date command on $SERVER running as $USER
my $spawn = Expect->spawn(qq{ssh $USER\@$SERVER date})
  or die "Unable to spawn ssh.\n";
$spawn->log_stdout(0);

# provide the password when prompted, waiting up to 5 seconds
if ($spawn->expect(5, 'password:')) {
    $spawn->send($PASS . "\n");
}
# wait for the date and print it out
if ($spawn->expect(5, -re => qr/^.*?\d{4}\r?\n/)) {
    print "The date on $SERVER is " . $spawn->match();
}
```

Example 3: get_date.pl *gets the date via ssh.*

dtSearch®
Instantly Search Gigabytes of Text Across a PC, Network, Intranet or Internet Site
Publish Large Document Collections to the Web or to CD/DVD
◆ over two dozen indexed, unindexed, fielded & full-text search options
◆ highlights hits in HTML, XML, & PDF while displaying embedded links, formatting & images
◆ converts other file types (word processor, database, spreadsheet, email, ZIP, Unicode, etc.) to HTML for display with highlighted hits
dtSearch DESKTOP with Spider $199
dtSearch WEB with Spider from $999
dtSearch Text Retrieval ENGINE for Win & .NET for Linux
dtSearch PUBLISH for CD/DVDs from $2,500
dtSearch NETWORK with Spider from $800
"Industrial-strength... superb"--PC Magazine
dtSearch Reviews...
◆ "The most powerful document search tool on the market" — Wired Magazine
◆ "Intuitive and austere ... a superb search tool" — PC World
◆ "Blindingly fast" — Computer Forensics: Incident Response Essentials
◆ "A powerful arsenal of search tools" — The New York Times
◆ "Covers all data sources ... powerful Web-based engines" — eWEEK
◆ "Searches at blazing speeds" — Computer Reseller News Test Center
The Smart Choice for Text Retrieval® since 1991
See www.dtsearch.com for:
◆ hundreds of developer case studies & reviews
◆ fully-functional evaluations
4 out of 5 of Fortune Magazine's most profitable companies purchased dtSearch developer or multi-user licenses in the past two years.
1-800-IT-FINDS
sales@dtsearch.com

date and prints it. When I run this script against my Redhat Linux 9 virtual machine (conveniently named Redhat9), I see:

```
$ ./get_date.pl
The date on Redhat9 is
                Thu Jun 17 08:41:24 EDT 2004
```

The VMPerl API includes an alternative mechanism for communicating with software running on the guest OS, using *set_guest_info()* and *get_guest_info()*. I did not use these methods for two reasons: First, they lack sufficient flexibility to control an interactive process easily. Second, to use them, you must install the VMWare Tools software on the guest operating system. This would add an extra step to the process of setting up a new machine, one that is otherwise unnecessary.

Putting It All Together

With the VMPerl API to start and stop machines and the Expect module to run commands via ssh, the rest is, as a former boss used to say, merely a simple matter of programming.

I started by designing a configuration file format to contain a description of all the machines in the farm. The format follows the same conventions as the Apache web server, made possible by the Config::ApacheFormat module created as part of the Krang project (http://www.cpan.org/authors/id/S/SA/SAMTREGAR/). Example 4 is a single machine's configuration from the file, farm.conf. Each machine in the farm gets a <Machine> block listing the username and password used to log in to the machine, a short description used in script output, and a list of all the Perl binaries on the machine paired with the Krang builds they generate.

One thing you might have expected to see in farm.conf that isn't there is the IP address of the machine. I decided to store that information in /etc/hosts because it is convenient to be able to ssh to the machine manually to debug problems. For example, the machine in Example 4 has a corresponding entry in /etc/hosts like this:

```
192.168.1.10 Redhat7_3_i686
```

With configuration out of the way, I created a class to encapsulate operations on the machines called KrangFarm::Machine. Example 5 shows a script that will start each machine on the farm and execute the *date* command. Notice how KrangFarm::Machine completely abstracts interaction with the VMPerl API and the farm configuration file.

```
# each machine gets a Machine block
<Machine Redhat7_3_i686>

   # a reminder of what's on this machine
   Description "Redhat 7.3 Server w/ custom Perls for i686"

   # the user and password the farm system will use to login,
     needs sudo
   User krang
   Password krang
   # the Perl binaries and the builds they generate
   Perls /usr/bin/perl             Redhat7_3-perl5.6.1-i686-linux \
         /usr/local/bin/perl5.6.2 Redhat7_3-perl5.6.2-i686-linux \
         /usr/local/bin/perl5.8.3 Redhat7_3-perl5.8.3-i686-linux \
         /usr/local/bin/perl5.8.4 Redhat7_3-perl5.8.4-i686-linux
</Machine>
```

Example 4: *A single machine's configuration block.*

```
#!/usr/bin/perl -w
use lib '/home/sam/krang-farm/lib';
use KrangFarm::Machine;

# loop through all configured machines
foreach $name (KrangFarm::Machine->list()) {
    $machine = KrangFarm::Machine->new(name => $name);
    $machine->start();

    # call the date command and extract the output
    $spawn = $machine->spawn(command => 'date');
    if ($spawn->expect(5, -re => qr/^.*?\d{4}\r?\n/)) {
        print "The date on $name is " . $spawn->match();
    }
    # stop the machine
    $machine->stop();
}
```

Example 5: *Script that starts each machine on the farm.*

The actual scripts in the Krang Farm system, krang_farm_build and krang_farm_test, aren't much more complicated than Example 5. The build script, krang_farm_build, transfers a source tar-ball to each machine, runs *make build* and *make dist* and fetches the resulting binary distribution. The test script transfers a binary distribution to each machine, installs it, runs *make test*, and parses the output to determine success or failure.

Building and testing on all configured platforms is as simple as:

```
krang_farm_build --
  from-cvs && krang_farm_test -- version 1.020
```

Plans for the Future

The Krang Farm is working well, but there's always room for improvement. Now that building and testing are automated, I plan to add a script to perform test runs automatically every night against the latest source in CVS. If the tests fail, the script will send mail to the Krang development mailing list.

Another possible enhancement would be a system to test upgrades from one version of Krang to another. Krang includes an upgrade facility that lets users move from one version of Krang to another without losing their data. Testing upgrades from an arbitrarily old version of Krang to the latest release is a time-consuming process, and automating it could help us find upgrade bugs faster.

This work may well be done by the time you read this article; drop by the Krang web site to find out, or even to lend a hand! Like Krang, the Krang Farm software is 100 percent open source.

Problems

For all the cheerleading in this article, the project wasn't without problems. One of Krang's supported platforms, Fedora Linux, isn't officially supported by VMWare. While Fedora Core 1 worked great, Fedora Core 2 ran extremely slowly under VMWare. Compiling Krang took around an hour versus 10 minutes on the rest of the machines. I eventually solved this problem by compiling a new kernel on the Fedora Core 2 machine with a lower value for HZ (100 versus the default of 1000), a tactic I found on the VMWare community message boards.

Additionally, I am unable to start machines as a nonroot user in the Virtual Machine Console. Judging by the documentation, I'm sure this is supposed to work, but I get a fatal error whenever I try it. However, starting machines using the VMPerl API as a nonroot user works fine.

DDJ

Jacl.NET

Porting a scripting language to .NET

WILL BALLARD

Jacl.NET is a port of the Jacl Java-based TCL interpreter to the .NET Framework. Jacl.NET gives .NET developers a TCL interpreter suitable for embedding within applications. In this article, I describe the issues I encountered in using Visual J# .NET to porting Jacl to .NET, and present an application of an embedded interpreter for database password management that eliminates the need to keep enterprise application passwords on disk. The complete source code for Jacl.NET is available at http://www.mailframe.net/ and from *DDJ* (see "Resource Center," page 5).

Jacl itself is a port of Sun's TCL interpreter from C to Java, combining core language and commands with Java-specific extensions. The port has existed in several revisions; the current version is 1.3.1 (http://sourceforge.net/projects/tcljava). However, unlike the C version of TCL, which supports a wide array of extensions such as OraTCL or Expect, Jacl addresses only the core TCL language. TCLJava, a second package intertwined with Jacl, provides the ability to script and control Java objects from TCL scripts running in the Jacl interpreter.

The Jacl port to J# .NET is intended to provide core TCL language scripting capability within .NET applications. As a scripting language, TCL can then be leveraged to add dynamic, interpreted functionality to .NET applications. The Jacl/TCLJava focus on allowing scripting of Java classes is removed and replaced with an emphasis of integration with the .NET environment, thereby leveraging the CLR and allowing multiple language development.

Porting Jacl to .NET lets you create TCL extensions using .NET, including making new TCL language commands in any .NET language. The port forms a platform to extend TCL for use within your own .NET applications as a command language, runtime expression interpreter, or script-based extensible application configuration system. Jacl in .NET provides anyone familiar with .NET and TCL the ability to extend the TCL language to suit their own application needs without the traditional C programming required to extend TCL.

Porting

The port began with the full Jacl 1.3.1 source tree taken directly from SourceForge. I created a J# project with the Jacl and TCLjava source directories included recursively to capture all Java language source files. I used J# to avoid porting the Java code to C#, making the best use of the existing Jacl code. The existing makefile was abandoned in favor of a .NET solution and project, since most .NET developers will use Visual Studio. The first pass of porting was simply getting the source to compile.

The first problem involved variable naming. Variables named *enum* are considered to be a keyword by the J# compiler in Visual Studio 2005. Prefixing these variables with the escape "@" character, turning them into *@enum*, lets them compile. Previous ports of Jacl using Visual Studio 2003 didn't require this, but in referring to the Visual Studio 2005 documentation for J#, the answer became clear—2005 adds enumerated type support to J# through the *enum* keyword.

The major effort in porting revolves around I/O processing. Versions of Jacl from 1.1.1 to 1.3.1 have different I/O abstractions. The most recent version (1.3.1) forms the major basis for the port making use of Java classes not available in .NET. The root of the porting problems occur in *TclInputStream,* which performs conversion with *sun.io.ByteToCharConverter.* Unfortunately, this class is not available in J#. Similarly, *TclOutputStream* performs conversion with *sun.io.CharToByteConverter.* Ultimately, the *TclInputStream* and *TclOutputStream* exist to have character and byte encoding streams available to the *Channel* class. This was introduced with Jacl 1.3.1 moving towards Java 1.2 capabilities. However, with J# at a level of Java substantially less than Java 1.2, and certainly not including Sun libraries, I merged the previous Jacl release implementation of *Channel* from the 1.1.1 code into the 1.3.1 code base.

> "Porting Jacl to .NET lets you create TCL extensions using .NET"

This I/O port activity affects the *FileChannel*, *StdChannel*, and *BgErrorMgr* classes in the Jacl port for pure I/O. *FblockedCmd* and *FConfigureCmd* are affected to be compatible with the simplified I/O Channel from prior to 1.3.1 Jacl. For logical I/O, *GetsCmd*, *OpenCmd*, *PutsCmd*, *ReadCmd*, and *SeekCmd* are ported from 1.1.1 instead of from 1.3.1 to

Will is an independent application architect based in Austin,Texas. You can contact him at wballard@mailframe.net.

work with the back version *Channel*. The final piece of I/O, *SocketCmd*, is also ported from 1.1.1. In summary, Jacl.NET I/O is not up to the Jacl 1.3.1 level. Still, for my needs (and testing), there were no negative consequences.

Jacl provides the capability to call and script Java objects through TCLJava. While useful in a Java setting, this is less useful within a .NET context. Consequently, the commands and support classes (Java*) that form the Java language object scripting capability are excluded from the .NET port solution. This streamlines the port to focus on TCL functionality, substantially reducing the size of the Java-specific porting effort. (Replacing this with the ability to script any .NET object interactively is planned as an extension.)

Resource streams in Java aren't really supported in J# in the same fashion. I converted these to use .NET assembly resources. This is important because Jacl uses TCL scripts embedded as resources to configure itself on startup. The embedded resource is a startup script that registers commands for execution and defines commands as TCL functions. Using assembly resources is an example of mixing and matching .NET classes with Java core classes directly into J# source. Even though Java I/O and streams are available, I chose to use the .NET streams for reading a resource (see Listing One), which reads an embedded resource script that was compiled into the assembly into a string for runtime evaluation. Notice how you can get a Type from a J# instance exactly like a C# instance. Except that I compile it as J#, this piece of code looks like — and might as well be — C#. J# is close to C# in language in many cases, with the differences expressed in library support. By leveraging the .NET classes, these differences can be minimized. Finally, I removed the mapping for the Java command handler in the interpreter main object *tcl.lang.Interp*. This associated the command string *jacloadjava* with a specific command object *tcl.lang.JaclLoadJavaCmd*.

These changes result in a core language TCL that compiles with J#. To fit in with the .NET console application model, I created a small console shell application that invokes the main interpreter class *tcl.lang.Shell*. Partly out of personal preference and partly out of wanting to demonstrate mixed-language development, this shell program (called "jaclsh") is created in C#. It is nearly a one-liner, directly invoking the Java-based main function passing the console arguments; see Listing Two. With the shell program in place, I ran the TCL test scripts included with Jacl. I removed the TCLJava-based tests — not supporting scripting Java objects, instead focusing on the tests for core TCL functionality.

In running tests, I found a series of small dependencies on the Java platform separate from the Java language — issues about paths, application names, and host system information. This led to changes in *tcl.langInfoCmd*, reminding me that Java is more than a language — it's also a platform. This is important in porting. There are issues with any port that are larger than just the code in the program, and you need to keep in mind how the code touches the environment and what the assumptions are for that code in its native environment. I had similar problems with the *clock* command because it depends on the environment variable system in Java. For instance, formatting time zones as GMT+00:00 instead of the expected GMT, as .NET shows time zones relative to GMT in string formats, instead of three-character time zone name abbreviations. Consequently, I completely replaced the *clock* command, converting it to use .NET-style dates, including *System.DateTime* parsing and formatting, replacing the custom date parsing, and processing in TCL. This was more to my liking, having become much more of a .NET programmer

.NET • COM • JAVA CHARTING SOLUTIONS

Need More Than A Pretty Face?

For Serious Enterprise-Level Data Visualization & Analysis

Multi-Platform
Ease of Use
Outstanding Support
...and A Pretty Face!

Extensibility
Performance
Scalability

Chart FX

©2004 Software FX. All rights reserved. Chart FX is a registered trademarks of Software FX, Inc. All other brands are owned by their respective owners.

than a Java programmer. This required that I expand support to include long integers to hold the nanosecond scale-tick values that *System.DateTime* uses, an unexpected piece of additional work, but beneficial for high-precision clock times.

Custom Commands

With a working .NET port, my next step was to extend the interpreter to provide custom commands. Essentially, all extensions to Jacl take the form of commands. TCL has a simplified, regular grammar with a mapping of a command word to a command class. Each command class is instantiated as an object and then takes an array of parameters, which may be literal strings or numbers, or the evaluated results of other commands. The commands themselves follow the classic command pattern, which creates an instance of a command class to serve as a verb or function, then invokes operations with an execute operation when you run a script. In the case of Jacl, the classes implement *tcl.lang .Command,* and the execute operation is called *cmdProc*. This object as function metaphor is prevalent in Jacl.

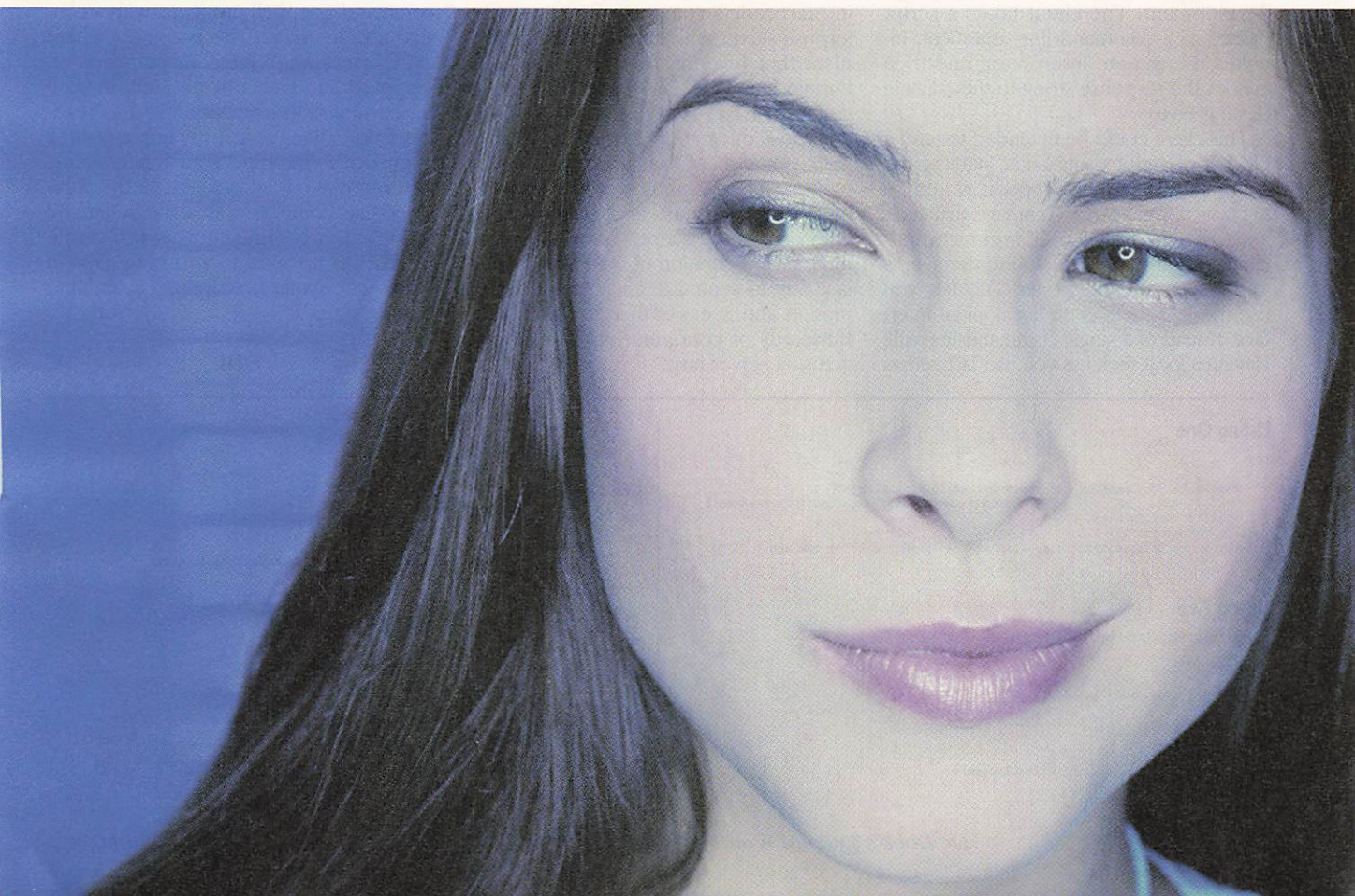
Major enterprise applications — database servers or even Windows itself — have remote command-line interfaces, such as isql for SQL Server, or netsh for Windows (among others). You can use Jacl.NET to easily create your own command-line control interface.

Practically every enterprise application accesses a database, and most of those store the connection information in either the registry or a configuration file. Many will even store the password in plain view on disk. Using Jacl.NET, a custom command, and an interpreter instance mounted with .NET remoting, you can create a means to configure applications at runtime without fixed configuration files left on the server. This lets you authenticate an enterprise server application to a database without having the password be part of the application itself or in any configuration file. As an example of command-line control, I have built a logon command (Listing Three) that tells a remote-process-running server where to connect for database access.

Creating a custom command is a straightforward implementation of the *tcl.lang.Command* interface. Jacl parses the arguments and parameters for you, passing a simple array of object parameters with the zero position being the name of the command. For logon, a single parameter *connection_string* suffices. It checks for the right number of parameters, and if they are in place, a connection to a database server is opened with the passed connection string parameter, and finally the connection string is saved in a static variable making it visible to the rest of the application. The command extension is implemented in C# to demonstrate mixed-language development. This has a C# class implementing an interface originally authored in Java. The *setResult* call is the output mechanism to allow a command to return information to the caller, as the *cmdProc* method itself has a *void* return type.

Remote Interpreter

With the command implemented, the next step is to make it accessible. This access is provided by a simple remoting server exposing the interpreter on a socket. While this is not the most secure option (remoting lacks transport encryption), it illustrates the point. In a secure setting, you could use an SSL socket for remoting or WSE 2.0 with WS-Security encryption to provide transport and some form of authentication and access control. However, remoting makes a simpler and shorter illustration of the remote scripting principle without including a lot of plumbing to expose the interpreter on the network.



InterpreterService registers the new custom command with a string that is the actual command text to type in TCL scripts in an interpreter instance; in this case, mapping the command string logon to the class *SampleServer.LogonCommand* via the *Extension* class method *loadOnDemand*. *Extension* serves as a registration hub for installing commands into an interpreter instance. The TCL interpreter uses the word to command mappings to locate and invoke command objects based on the passed input script.

The interpreter instance in this remote server is accessed by a single remote method that calls *eval* to run the script text. A class inheriting from *MarshalByRefObject* wraps the interpreter and provides an input method to submit scripts. On a successful run, the output of the script run is returned as a string. Exceptions are propagated automatically by remoting as long as they are serializable!

The service is exposed on a *TcpChannel* as a well-known object allowing lookup by URL. The client uses a remoting URL to find a server to run scripts; see Listing Four.

The remote client is straightforward, connecting to a published remote instance of the *InterpreterService* before invoking a script passed as a command-line argument. I implemented this client in VB.NET, continuing the theme of mixed-language development. The client takes a script passed as a command-line argument, invokes the remote interpreter, and then echoes the response string to the console (Listing Five).

This client could be extended to read a file full of script commands, passing the entire content. Another option would be to create a remote TCL command that takes a set of scripts as a TCL argument in a client-side interpreter, passing the scripts to a server-side interpreter. This lets you use the Jacl shell as the local client interface instead of a single command-line call. Having a local shell lets you use TCL scripts to coordinate multiple commands from the client to a single server, or send commands to multiple servers each exposing the remote interpreter service.

To invoke Jacl.NET, you first run the remote interpreter server; for example:

```
SampleServer.exe
press enter to exit...
```

With the remote interpreter running, you issue a command at the prompt to check the error handling:

```
SampleClient.exe
        "logon {bad connection string}"
```

In this example, the format of the initialization string does not conform to specification starting at index 0. Then you actually connect:

```
SampleClient.exe
   "logon {Server=DOOZEY;user=
     XXX;password=XXX;Database=XXX}"
```

which echoes back the success message:

```
Connected
```

This simple scenario illustrates all the basic principles in creating your own commands — implement the *tcl.lang.Command* interface, create an *Interp*, register with *Extension*, and make it available to call.

This example also illustrates the point of remotely altering the running state of a server with script. Using such a system in practice lets you control a custom enterprise server with an extensible mechanism that leverages an existing, well-known language. The benefit here is in control. I have used this technique with enterprise integration projects to get fine-grained control of the startup sequence of multiple interoperating services and to remotely manage clusters of similar servers with predefined scripts. Allowing the remote reconfiguration of servers at runtime is a way to centralize and control configuration and is easier than trying to keep hundreds of configuration files in sync across a server farm.

Conclusion

There are a number of improvements that can be made to Jacl.NET. For instance, adding .NET language object scripting capabilities, much like TCLJava, would be useful, allowing interpreted prototyping similar to other scripting projects such as IronPython (http://www.ironpython.com/). Such commands could be effectively implemented with reflection. This feature is a planned extension, based on demand. To date, the primary use of Jacl.NET has been for embedding within applications, not scripting objects.

Sticking with configuration and control, more than just database connections can be configured. Keeping a simple configuration class with dynamic settings, for example adding a hash table to the logon command to allow storing more than one setting, enables a system of hot configuration changes, letting you access and modify running server settings without a restart to pick up an application configuration file.

TCL in your application can give you capability for evaluated, user-defined fields, and formulas or customizable event-handling scripts. Lots of little pieces of math can be delegated and interpreted easily and efficiently to make fairly sophisticated algebraic calculators within your programs.

The command-line interpreter in Jacl forms the basis for creating your own command-line shell programs with a console user interface. This is an area where .NET is lacking. With all its emphasis on graphical and web UIs, the built-in shell processing is essentially limited to reading/writing lines of text. By implementing commands wrapping your functionality, you can make it all available at the command line, complete with a line buffer and inline editing similar to *cmd*. Overall, Jacl.NET is a useful tool to plug gaps in .NET with dynamic scripting capability.

DDJ

Listing One

```
try
{
    System.IO.StreamReader rdr = new System.IO.StreamReader(
            this.GetType().get_Assembly().GetManifestResourceStream(resName));
    eval(rdr.ReadToEnd(), 0);
} catch (Exception e2) {
    throw new TclException(this, "cannot read resource \"" + resName + "\"");
}
```

Listing Two

```
static void Main(string[] args)
{
    try
    {
        tcl.lang.Shell.main(args);
    }
    catch (Exception ex)

    {
        System.Console.Error.WriteLine(ex);
    }
}
```

Listing Three

```
public class LogonCommand : tcl.lang.Command
{
    /// <summary>
    /// This holds the connection string once a connection is made.
    /// </summary>
    public static string ConnectionString;
    #region Command Members
    public void cmdProc(tcl.lang.Interp interp, tcl.lang.TclObject[] objv)
    {
        //check argument length
        if (objv.Length == 2) //enough command arguments
        {
            try
            {
                System.Data.SqlClient.SqlConnection conn =
                  new System.Data.SqlClient.SqlConnection(objv[1].ToString());
                conn.Open();
                ConnectionString = conn.ConnectionString;
```

```
                conn.Close();
                interp.setResult("Connected");
            }
            catch (Exception ex)
            {
                interp.setResult(ex.Message);
            }
        }
        else
        {
            //standard message for missing arguments
            throw new tcl.lang.TclNumArgsException(interp, 1,
                                                   objv, "connection_string");
        }
    }
    #endregion
}
```

Listing Four

```
class Program
{
    static void Main(string[] args)
    {
        InterpreterService service = new InterpreterService();
        TcpChannel channel = new TcpChannel(55000);
        ChannelServices.RegisterChannel(channel);
        ObjRef mounted = RemotingServices.Marshal(service,"interpreter");
        System.Console.WriteLine("press enter to exit...");
        System.Console.ReadLine();
    }
}

/// <summary>
/// Expose the <see cref="tcl.lang.Interp"/> via remoting.
/// </summary>
public class InterpreterService : MarshalByRefObject
{
    private tcl.lang.Interp _interpreter = new tcl.lang.Interp();

    public InterpreterService()
    {
        tcl.lang.Extension.loadOnDemand(_interpreter, "logon",
                                        "SampleServer.LogonCommand");
    }
    /// <summary>
    /// Execute a script, returning the evaluated results.
    /// </summary>
    /// <param name="script"></param>
    /// <returns></returns>
    public string RunScript(string script)
    {
        try
        {
            _interpreter.eval(script);
            return _interpreter.getResult().ToString();
        }
        catch (Exception ex)
        {
            return ex.ToString();
        }
    }
}
```

Listing Five

```
Module ClientEntryPoint
Sub Main(ByVal args As String())
    If (args.Length > 0) Then
        Dim chan As New TcpChannel()
        ChannelServices.RegisterChannel(chan)
        RemotingConfiguration.RegisterWellKnownClientType(GetType
                    (SampleServer.InterpreterService),
                    "tcp://localhost:55000/interpreter")
        Dim remote As New SampleServer.InterpreterService()
        System.Console.WriteLine(remote.RunScript(args(0)))
    End If
End Sub

End Module
```

DDJ

More .NET on DDJ.com

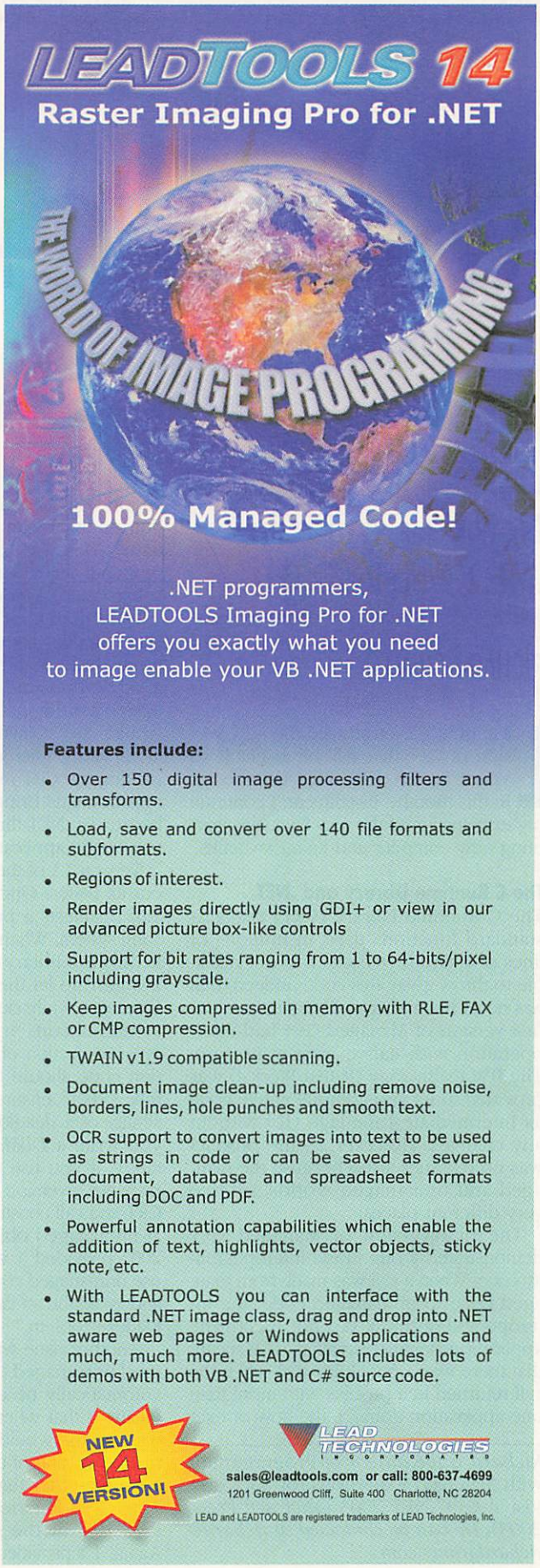
ASP.NET2theMax: One-Click Attacks

Hackers can trick unsuspecting e-mail users into launching attacks on your site with just a single click. Here's how you can filter out the bogus requests from the legitimate ones.

Grimes on .NET: Reading .NET Performance Counters

Here's a look inside the .NET classes that you can use to read counters provided by other processes.

Available online at http://www.ddj.com/topics/dotnet/


LEADTOOLS 14
Raster Imaging Pro for .NET
THE WORLD OF IMAGE PROGRAMMING
100% Managed Code!
.NET programmers,
LEADTOOLS Imaging Pro for .NET
offers you exactly what you need
to image enable your VB .NET applications.
Features include:
• Over 150 digital image processing filters and transforms.
• Load, save and convert over 140 file formats and subformats.
• Regions of interest.
• Render images directly using GDI+ or view in our advanced picture box-like controls
• Support for bit rates ranging from 1 to 64-bits/pixel including grayscale.
• Keep images compressed in memory with RLE, FAX or CMP compression.
• TWAIN v1.9 compatible scanning.
• Document image clean-up including remove noise, borders, lines, hole punches and smooth text.
• OCR support to convert images into text to be used as strings in code or can be saved as several document, database and spreadsheet formats including DOC and PDF.
• Powerful annotation capabilities which enable the addition of text, highlights, vector objects, sticky note, etc.
• With LEADTOOLS you can interface with the standard .NET image class, drag and drop into .NET aware web pages or Windows applications and much, much more. LEADTOOLS includes lots of demos with both VB .NET and C# source code.
NEW 14 VERSION!
LEAD TECHNOLOGIES INCORPORATED
sales@leadtools.com or call: 800-637-4699
1201 Greenwood Cliff, Suite 400 Charlotte, NC 28204
LEAD and LEADTOOLS are registered trademarks of LEAD Technologies, Inc.

Whidbey C++ Syntax

New C++ syntax in .NET Version 2.0

RICHARD GRIMES

Last month, I introduced the new version of C++ that will be available in Visual Studio .NET 2005 (Whidbey). In this month's installment, I continue my examination of the new language, covering new compiler and linker switches.

The C Runtime Library and .NET

The C Runtime Library (CRT) contains standard functions used in millions of lines of C and C++ code. It is not possible to throw away this code; indeed, that has never been Microsoft's intention. The first version of Managed C++ had interoperation with native code at its core: The IJW technology (It Just Works!) was provided to let you compile native code as Intermediate Language (IL) without any changes. However, as the product was used, issues arose because the managed and unmanaged worlds are two very different places.

The first version of Managed C++ demonstrated a bug in the interoperability code: When a call was made from managed code to unmanaged code, the interoperability thunk did not record the application domain identifier. This meant that there was a possibility that when the call returned in a process with more than one application domain, the flow of execution could continue in an application domain other than where the call was initiated. This was fixed in 1.1 of the runtime, but interoperability is fraught with subtle issues. Application domains present a problem to the CRT because it's a concept that does not exist in the unmanaged world. Indeed, the CRT was designed for a single-threaded world—UNIX developers would *fork* a new process that would have its own copy of the CRT global variables. Microsoft did a good job of making the CRT multithreaded, but .NET app domains bring up another problem because a .NET thread is not constrained to a single application domain.

Version 1.1 of the runtime was not without its issues. One issue (also present in 1.0) manifests a problem with the operating system. When Windows loads a DLL, it calls the library's entry point (usually *DllMain*) to let the library initialize itself. However, while doing this, the operating system obtains and holds onto various synchronization objects, so the initialization code should be restricted to only a few APIs to ensure that there is no possibility of a deadlock. Managed C++ lets you write a *DllMain*, but more importantly, if you use the CRT, the compiler provides a version for you to initialize the CRT and call constructors of static or global unmanaged objects. Visual C++ .NET 2003 provided a workaround to let you use unmanaged code in managed libraries and avoid the so-called "DLL mixed-mode loader problem."

Another issue with C++ was that if you used unmanaged code, the library would automatically be unverifiable. What this means is that when the library is loaded, the runtime would give the library minimal permissions if the assembly came from a source other than the local hard disk. Thus, C++ libraries could not be downloaded over the Internet or intranets. Again, 1.1 provided a workaround for this issue to make it more likely that your library would be verifiable; however, it was messy to implement.

Mixed-Mode Assemblies

The Whidbey compiler recognizes these issues and provides switches to take them into account. The */clr* switch has three options: */clr*, */clr:pure*, and */clr:safe*. The idea is that you determine the verifiability requirement of the assembly and also the details about how unmanaged code is treated, then the compiler generates appropriate code and, in some cases, issues errors if the code does not meet the criteria. This means that it is no longer necessary to use the workarounds created for .NET 1.1 to get around the loader lock bug, or to create a verifiable assembly.

> "Mixed-mode assemblies contain x86, unmanaged code"

The first option, */clr*, is for so-called "mixed-mode" modules—those that can contain unmanaged code compiled as x86; so by definition, the module is not verifiable. When you compile with this switch, it is assumed that you will use the CRT and so the compiler links with msvcmrt.lib (an import library for msvcm80.dll).

(continued on page S9)

Richard is the author of Programming with Managed Extensions for Microsoft Visual C++ .NET 2003 *(Microsoft Press, 2003). He can be contacted at richard@richardgrimes.com.*

Microsoft®, MSDN Magazine, and Tech Conferences
join forces to produce the premier event for Microsoft developers in 2005.
It will impact how you build solutions, gain productivity, and enhance your development skills to give your company the competitive edge!

CONNECT TO THE SOURCE!

March 20-23, 2005
Orlando, Florida
Hyatt Regency Grand Cypress Resort

SCOTT GUTHRIE
PRODUCT UNIT MANAGER, ASP.NET

MARIE HUWE
GM OF THE DEVELOPER TOOLS, MARKETING

ARI BIXHORN
LEAD PRODUCT MANAGER, WEB SERVICES

TOM RIZZO
DIRECTOR, SQL SERVER PRODUCT MANAGEMENT

MICROSOFT ASP.NET CONNECTIONS

Visual Basic .NET ⟷ C#
Visual Studio CONNECTIONS

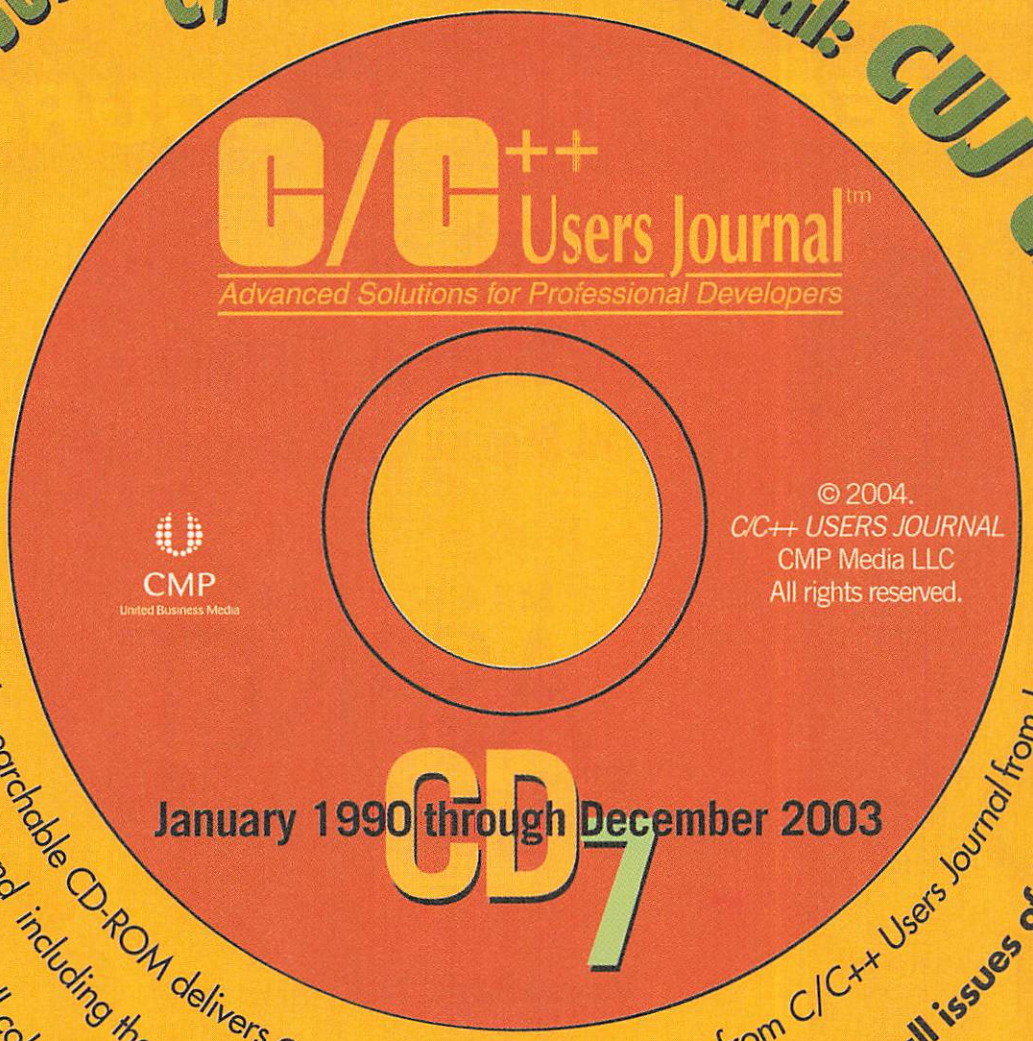
REGISTER EARLY FOR ONE EVENT and attend the sessions of the concurrently run events for **FREE**, including SQL Server Magazine Connections.

SQL Server magazine CONNECTIONS

Register Today! www.DevConnections.com 800.438.6720

BROUGHT TO YOU BY: Microsoft msdn magazine TECH Conferences Inc. PENTON MEDIA

Just Released from C/C++ Users Journal: CUJ CD - 7



This fully searchable CD-ROM delivers **over 165 issues** of content from C/C++ Users Journal from January 1990 through, and including the December 2003 issue of C/C++ Users Journal, **plus all issues of the Java Solution Supplement** and all columns of CUJ's online C++ Experts Forum. Save time, just cut and paste information right into your work in progress, **24/7**, without worrying about long download times.

C/C++ Users Journal™
Advanced Solutions for Professional Developers

CMP
United Business Media

To Order:

Online: www.ddj.com/cdrom/
US Toll-Free: 1-800-444-4881
International: 1-785-838-7500
Fax: 1-785-838-7566

(continued from page S6)
In addition, the compiler adds *COMIMAGE_FLAGS_32BITREQUIRED* to the *.corflags* member of the CLR header in the assembly to indicate that the assembly must be run in a 32-bit process because the assembly contains function fixups. A fixup is essentially an unmanaged function pointer and is determined at runtime; for the 32-bit compiler, the fixup is always 32 bits!

Using unmanaged libraries is a problem because unmanaged libraries have no concept of .NET application domains. In native code, the module entry point is used to initialize global variables and call constructors of static objects. However, this can cause a problem if the constructors of these objects call code that is not safe in *DllMain*. To get around this, the Whidbey compiler adds a module static constructor to perform initialization. In .NET, a static constructor is guaranteed to run before the first use of the constructor's type. By creating a module static constructor, the initialization code is called sometime after the library's entry point has been called— after any .NET initialization has occurred, but before any managed code is executed.

Consequently, if you look at an assembly compiled with */clr* using ILDASM, you see a static method called *.cctor* and various other static methods used to access the application domain and initialize the native code. Many of these methods are native; hence, the compiler marks the assembly with *SecurityPermissionFlag.SkipVerification* to indicate that the assembly can be loaded only if it has this permission (which usually means that the assembly is installed on the local hard disk). By default, in mixed mode, all global variables are global to all application domains in the process; that is, one single copy of the data exists regardless of the application domain used to access the data. If you want a native, global object to be global to only the application domain (in effect, a copy of the object for each application domain), then you can use the *__declspec(appdomain)* modifier.

When you use the */clr* compiler switch, your code is automatically linked with the msvcmrt.lib static import library. You cannot use the statically linked CRT (the */MT* or */MTd* options) with */clr*.

The C++ compiler can be used to create .NET modules, and previous versions of the compiler used the */clr:noassembly* switch. However, in the new version of the compiler, there are three types of code generated (according to its verifiability and whether x86 code is permitted) and */clr* is used to determine the type of code generated. Because of this, the */clr:noassembly* switch has been replaced with */LN*, which is used to produce a .NET module. In addition, the compiler expects the C++ code to use the new language syntax by default. If you want to compile a source file that uses the old managed C++ syntax, then you can use the */clr:oldSyntax* switch. This means, of course, that the code is compiled as mixed mode.

Pure Assemblies

Mixed-mode assemblies contain x86, unmanaged code. By definition, this is not verifiable. Mixed-mode assemblies are great if you want to use existing native C++ static libraries, a native .obj file, or native code that does not compile to IL. There are two other options for compiling an assembly— pure and safe. Both types only contain IL and contain no x86 code. However, of these two, only "safe" is verifiable code.

You use */clr:pure* if you want to use unmanaged types in your code that are included through source files that are compiled by the managed compiler; for example, template files. The native code is compiled to nonreference types; that is, they are not created on the managed heap, but still compiled to IL and run under the .NET runtime (this is achieved because the types are essentially compiled as value types). However, because these types are not managed, it means that the code is not verifiable.

Again, static and global unmanaged objects have to be initialized before they are used, so pure assemblies also have a module static constructor. The difference between the two is that pure assemblies maintain a copy of each global object for each application domain. If you prefer a global native object to have a single value for all application domains in the process, you can mark the variable with the *__declspec(process)* modifier. Pure assemblies can contain native C++ classes, but not x86 code. This still means that the assembly is not verifiable; hence, a pure assembly is marked as requiring the *SkipVerification* permission.

Pure assemblies can call native code exported from a DLL through platform invoke. The CRT is provided through the msvcm80.dll DLL, and a pure assembly has an external module reference to this in its manifest. The linker accesses the CRT methods through the msvcurt.lib import library. Whereas mixed assemblies statically link CRT functions to the assembly as embedded x86 code, pure assemblies access the CRT functions exported from msvcm80.dll through platform invoke. This way, pure assemblies can ensure that they only contain IL. Indeed, if you look at the manifest using ILDASM,

Bar Code ActiveX and DLLs
Easily add unlimited professional bar code generation capabilities to Windows and web applications.
UPC-A & E
PDF417
Code 128
Code 39
And Many More...

FREE DEMO!
TAL tech www.taltech.com 1-800-722-6004

Just Released and Now Available

This new release gives 24/7 access to source code, algorithms, programming paradigms, columns, and much more from 16 years — ***YES, 16 YEARS*** and over 190 issues of ***Dr. Dobb's Journal!***

No more taking time to search through issues of *DDJ* to find information. Just cut and paste right into your programming project. No bandwidth issues either! Information is just a click away on your CD Drive — 24/7!

ORDER TODAY!
www.ddj.com/cdrom/

Dr. Dobb's CD-ROM LIBRARY

you'll see that a pure assembly has a *.corflags* value of *COMIMAGE_FLAGS_32BITREQUIRED | COMIMAGE_FLAGS_ILONLY*; that is, it only contains IL, but will call functions through platform invoke and so must be run in a 32-bit process.

Safe Assemblies

The final type of assembly you can create is a verifiable assembly, created with */clr:safe*. This is the most restrictive type from a C++ point of view because you cannot have native types, x86 code generated from your code or linked from a static link library, nor can you access any native code through a platform invoke. The advantage of code compiled with */clr:safe* is that it is verifiable; that is, the compiler performs a verifiable check and issues errors if any of the code is not verifiable. Verifiability will be an important issue in the future. For now, it appears only to be a security check to make sure that code that you have downloaded from the Internet does not try anything out of the ordinary. However, as .NET code finds its way into the operating system or enterprise services, such as SQL Server, then being verifiable as being safe code starts to become an important issue. Visual C++ provides this functionality in Whidbey—ready for the operating systems and services of the future.

A verifiable assembly can only call components in other assemblies that are also verifiable, even though msvcm80.dll is compiled as a .NET library assembly and still contains embedded x86; hence, the assembly demands the *SkipVerification* permission. So safe code cannot use the CRT at all.

> "A verifiable assembly can only call components in other assemblies that are also verifiable"

Linker Switches

The compiler switches can be used to compile individual source files without linking using */c*, then an assembly is created by linking the .obj files together. The overall verifiability of an assembly is determined by the lowest verifiability of the individual .obj files that are linked together. If you decide that you want a lower level of verifiability, then you can specify this using the */CLRIMAGETYPE* linker switch. The options are IJW, PURE, and SAFE, which correspond to mixed mode, pure, and safe.

Win32 code does not use exceptions to report problems because few languages could handle structured exception handling (SEH) exceptions. (There are a few Win32 APIs that generate SEH exceptions, but only if specific flags are passed to the function.) Instead, Win32 functions indicate that an error occurred through a return value (usually a *FALSE* value) and the caller can obtain details about the error by calling *GetLastError* to get a 32-bit value. Platform invoke has always had the ability to obtain such error codes, but it was an optional facility. The managed code calling the native function would use the *[DllImport]* attribute and set the *SetLastError* field to *true*. Then, when an error occurs, the runtime calls *GetLastError* and caches the result, so your managed code would then have to call *Marshal.GetLastWin32Error* to get the error code.

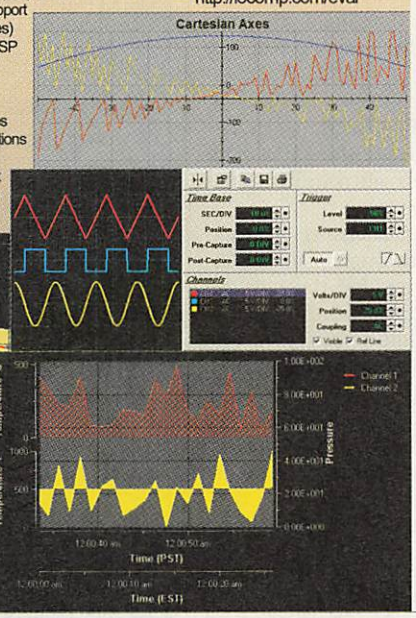
Real-Time Chart, Plot, & Scope Components

- High-Speed for Real-Time Applications
- Professional and Easy to Setup with Custom Property Editors
- Industry Standard OPC(OLE for Process Control) Client Built-In
- Includes iPlot, iXYPlot, and iScope Components
- Analog/Digital Real-Time Oscilloscope
- Unlimited Channels, Axes, Limits, Annotations, Cursors, Tables
- Visual Layout Manager (Design-Time and Run-Time)
- Value/Exponent/Prefix/Date-Time/Price 32nds Scale Labels
- User Can Scroll, Zoom, or Cursor Even While Data is Plotting
- Intelligent Auto Scale Labeling and Sliding Scales
- Curve Fitting (Cubic Spline, Polynomial, Rational)
- Run-Time User Toolbar and Custom Property Editor
- Copy, Save, Print, and Print Preview Support
- Log Files (Easily exports tab delimited files)
- EMF, BMP, JPG, and PNG Export for ASP
- X-Axis and Y-Axis Rotation
- X-Axis and Y-Axis Stacking
- Data Drill Down
- Reversible Linear and Logarithmic Scales
- Royalty Free for Distribution with Applications
- Custom Channel Data Point Markers
- Null Data and Empty Data Point Support
- Translation Support (Internationalization)
- Printed 230+ Page Manual Available

Plot Pack
ActiveX/VCL-CLX Native: $695.00
.NET Native: $849.00
Color Printed Manual: $75.00

Ultra Pack [Plot Pack and Instrumentation Pack Pro]
ActiveX/VCL-CLX Native: $1,395.00
.NET Native: $1,699.00

Free Evaluation Download
http://iocomp.com/eval

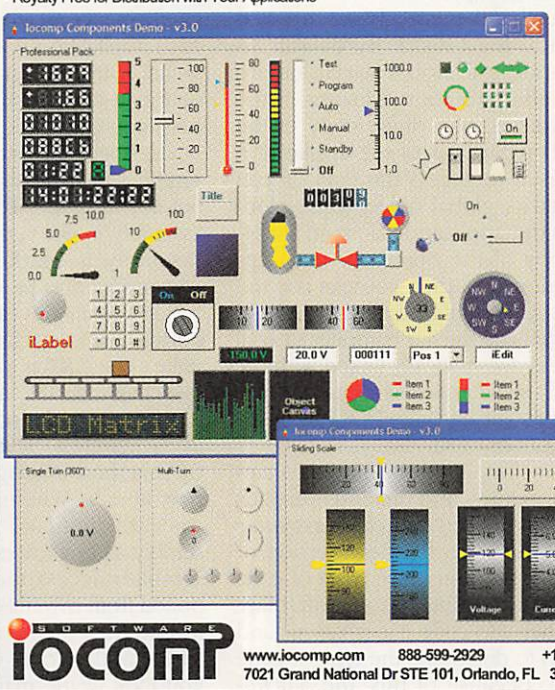


Industrial HMI and Scientific Components

- High-Speed for Real-Time Applications
- Professional and Easy to Setup with Custom Property Editors
- Industry Standard OPC (OLE for Process Control) Client Built-In
- Automatic and Custom Sizing, No Restrictive Bitmaps
- Look and Feel of real instrumentation hardware
- EMF, BMP, JPG, and PNG export for ASP
- Free Technical Support and Updates
- Royalty Free for Distribution with Your Applications

Instrumentation Pack Pro
ActiveX/VCL-CLX Native: $895.00
.NET Native: $1,099.00

Instrumentation Pack Std
ActiveX/VCL-CLX Native: $449.00
.NET Native: $559.00



60+ Components

Instrumentation Real Time Components

Full Source Code Available

SOFTWARE iocomp

www.iocomp.com 888-599-2929 +1-407-226-3456
7021 Grand National Dr STE 101, Orlando, FL 32819 USA

The new linker has a switch called */CLRSUPPORTLASTERROR*. When used without a parameter, this switch means that all calls through platform invoke will be accompanied by a call to *GetLastError*, regardless of whether the *SetLastError* field is set, and the result is cached for a subsequent call to *Marshal.GetLastWin32Error*. Clearly, this is a performance issue, so you can use the *SYSTEMDLL* option to restrict this behavior to just system DLLs. Finally, you can use the *NO* option to turn off this facility.

The linker also provides the switch */CLRTHREADATTRIBUTE*, which indicates the default COM apartment attribute of the entry point. This attribute is only used if the code calls a COM object (directly or indirectly). Since every COM object must be run in an apartment, the runtime makes the thread join a COM apartment just before it makes the first call to COM on the thread. By default, the runtime makes the thread join the process's multithreaded apartment (MTA), but if the COM object is marked as being single threaded, then the COM object is created in an STA apartment and the current thread uses COM marshaling to access the object. Even worse, if the thread is the GUI thread in a Windows Forms application, then the thread must join an STA, because if it enters the MTA, it can cause the UI to become unresponsive.

In earlier versions of the runtime, you used the *[STAThread]* or *[MTAThread]* attributes on the entry point to indicate the apartment that you would prefer the main thread to join. If you use one of these attributes, then the value supplied through */CLRTHREADATTRIBUTE* is ignored. Otherwise, the switch lets you specify the apartment for the main thread of the process (but not for other threads, where the default will still be MTA).

When you compile a mixed mode or a pure assembly, you do it so that you can use native C++ code in your assembly. Such assemblies will use the CRT and (for mixed assemblies) this might be embedded x86 code or (for mixed and pure assemblies) it will be a platform invoke call to the msvcm80.dll library. Even though this library is managed and shared, it is not installed in the GAC. Instead, it is installed as a shared Win32 library in the WinSxS folder (a so-called side-by-side assembly). When your code uses a side-by-side assembly, it declares the version of the assembly in a manifest file. This is an XML file with a manifest extension. By default, the linker will generate a manifest file for you for mixed and pure assemblies, but if you do not want it to do this you can use the */MANIFEST:NO* switch. The default name for a manifest file is the name of the assembly suffixed with *.manifest*, but you can change this name with the */MANIFESTFILE* switch. By default, the linker adds the msvcm80.dll library to the manifest file's dependency list, but you can add other entries using the */MANIFESTDEPENDENCY* switch.

```
#pragma comment(lib, "user32.lib")
extern "C" int __stdcall MessageBeep(unsigned int);
void main()
{
   MessageBeep(0);
}
```

Example 1: *Compile this code for mixed mode and pure mode and compare the IL and IAT entries that the compiler generates.*

Platform Invoke

One of the problems that the new version of the compiler has sought to solve is the so-called "double platform invoke problem." When you create a native function in your code, the compiler creates two entry points — one managed and the other unmanaged. The unmanaged entry point for the function provides a thunk that performs the necessary transition between unmanaged and managed worlds. It is used when a pointer to the function is passed to unmanaged code or is called by code that cannot be compiled to IL (for example, it has *varargs* or if the function is compiled with *#pragma unmanaged*). In some situations (*vtables*, for example), the entry point must be obtained at runtime. But this creates a dilemma for the compiler, because at compile time there is not enough information to determine whether the code requiring the entry point is managed or native. Consequently, the compiler for Version 1.1 of the runtime chose the unmanaged entry point. This meant that if the code requiring the entry point was managed, it would get an unmanaged entry point; hence, there would be another transition from the managed to the unmanaged world. This is the source of the name of the problem because a call through the managed *vtable* entry would require a transition from the managed world to the unmanaged world to the unmanaged entry point, and then from the unmanaged world to the managed world to get to the actual function: a double transition.

The new version of the compiler provides a solution to this issue. The *__declspec(__clrcall)* can be used on methods in mixed and pure code to indicate that the method should not have an unmanaged entry point. Because the method will only have a managed entry point, it means that it can only be called by managed code, but it also means that the call will always be through the managed entry point. It's worth pointing out that because pure and safe code do not have to accommodate embedded x86 code, they do not suffer from the double platform invoke problem.

The final topic I cover also concerns platform invoke, namely calling native code. Earlier versions of Managed C++ could use platform invoke through the *[DllImport]* attribute, but this assumed that the code had been rewritten as managed C++. If you used existing C++ code and compiled it with the */clr*, then the code would call DLL exported code through a static import library. In effect, this would add a thunk from the managed world to the unmanaged call to the DLL exported function. The thunk would effectively wrap a *JMP* to a memory location in the import address table (IAT) that would be fixed up at runtime. This is essentially how calls to DLLs are made in native code. If you ran *dumpbin /imports* on such an assembly, you would find that the IAT would contain an entry for the DLL functions you called.

In Whidbey, this behavior still occurs for mixed-mode assemblies. However, when you compile pure code, such embedded native code is not allowed and the compiler replaces the call with a *platform invoke* call. The code still looks as if it has a thunk because it still links to the DLL's import library. However, a call to *dumpbin /imports* shows that the IAT is empty. To see the effect of this, compile Example 1 with */clr*, then with */clr:pure*, and for both assemblies, use *dumpbin /imports* to show the IAT. Furthermore, use ILDASM and examine the method the compiler generates for *MessageBeep*: For mixed code, this method has the *pinvokeimpl(lasterr stdcall)* modifier indicating that a thunk is created, whereas the pure assembly has the *pinvokeimpl("USER32.dll" lasterr stdcall)* modifier indicating that platform invoke is used directly.

Next month, in the final installment of this series about the new version of Managed C++, I'll examine new features in the language: generics, array declarations, and the new pointer operators.

DDJ

Macros, Automated Builds, & Visual Studio 2003

Making use of an extensible object model

JAKE WATKINS

Visual Studio 2003 has an object model that encourages you to extend its capabilities. You can access the objects three ways—by writing add-ins or plug-in modules, using it as a component within external applications, or by writing Visual Basic macros. Each approach has advantages. In this article, I focus on writing macros, which are an excellent way to automate repetitive tasks. For instance, maintaining NANT build scripts is one repetitive task that can be automated using macros (for a description of NANT, see the accompanying sidebar entitled "A Brief Description Of NANT").

Within the Visual Studio object model is a set of objects that represent solutions, projects, files, and settings that make up a Visual Studio solution. When you execute the build command within Visual Studio, it uses this information to build your solution. Here, I present a macro that uses NANT to drive the build instead. The macro generates a NANT script and compiles a C# project. (You can add support for Visual Basic, too.) A second macro generates a skeleton master build file that calls each of the project builds.

While automated builds make the development process more manageable, maintaining an automated build is still a complex task. To make the build easier to manage, each project is put in its own build file. A master-build file is then generated separately to drive the build. This approach makes it easier for you to add extra steps to the build process. For instance, you might want to include a set of unit tests in your build. With this approach, you can rerun the macro to regenerate the project files and not lose any of your customizations in the master-build file.

Jake is a programmer for BORN in Dallas, Texas. He can be contacted at jakew@guerillaprogrammer.com.

To begin writing the macro, first start Visual Studio. To make debugging the macro easier, open or create a project. Next, open the macro explorer by going to the Other Windows submenu under the View Menu. (Alternatively, you can just press Alt+F8.) The macro explorer is typically docked with your solution explorer and class viewer. You should see two nodes in the macro tree—MyMacros and Samples. Both are provided by Visual Studio when you install it. Expand MyMacros and you see "Module1" appear below it. Right click on Module1 and choose New Macro. This start-up uses the Visual Studio Macros editor, which is another instance of Visual Studio, but it has your MyMacros file open in it.

All Visual Studio macros have to be written in Visual Basic, which provides full access to the .NET Framework. The macro file already has two imports at the top.

- *EnvDTE* is the Visual Studio object model that I will be using.
- *System.Diagnostics* provides classes to assist with debugging; for instance, the Debug Assert method lives here.

To write the macro, you need to import a few more namespaces. First, you need the *VSLangProj,* which provides a set of objects that extend the Visual Studio object model specifically for C# and Visual Basic projects. Next, you need the *System.IO* namespace to read/write files. *System.Text* is needed for string builder, and the *System.XML* namespace to generate the NANT XML build script.

I divided the macro into separate setup subroutines, making it easier to modify for other purposes. The first subroutine iterates over the collection of projects in the solution. For each C# project it finds, it generates the project's build script. To iterate over the collection of projects, you first have to get references to the collection of projects in the active solution. This collection follows normal Visual Basic practices as its first index is at 1 (I expected it to be at 0, which caused me some frustration initially). Iterate over the collection and call a subroutine for each C# project; see Listing One. The *VSLangProj* namespace provided the *prjKind* object used in the *if* statement. Using the *prjKind* object makes the code more readable. I would have had to compare against the C# project GUID without it.

> "Automated builds make the development process more manageable"

If you have a C# project, call the *GenerateCSharpProjectBuild* subroutine and pass it the project object to work on. To include Visual Basic, you would also need to generate *vb* tasks. A simple *csc* task looks something like Listing Two.

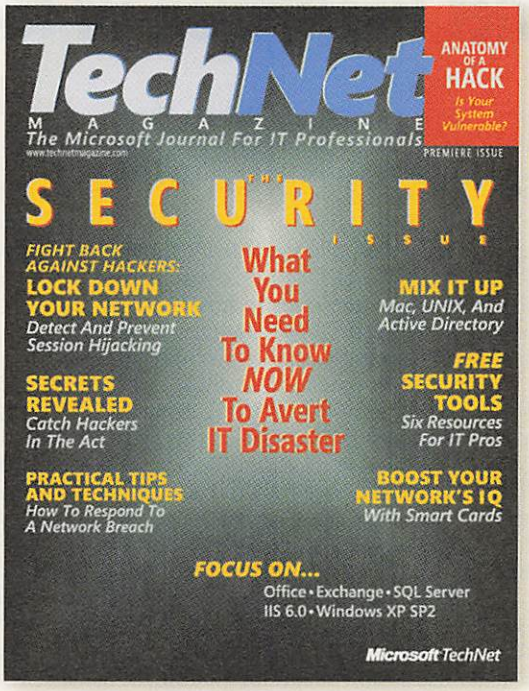
To make generating the build for the C# project easier, I split the process by first setting up the build file and generating the basic properties, then dealing with references, resources, and source files in separate subroutines. The properties for the first step are the output directory, the name of the output assembly, and whether you are compiling for release or debug (Listing Three).

The build file is kept in the same directory as the project file; the directory name comes from the project's properties collection. The properties collection is a

FREE!

For qualified subscribers in the U.S. only

TechNet Magazine



TechNet Magazine, a joint publication of Microsoft Corporation and CMP Media LLC, is offering a special edition to qualified IT professionals.

TechNet Magazine will provide in-depth, hands-on technical information that IT professionals can use in their daily work, covering the most current solutions for managing IIS, Exchange Server, Active Directory, Office, Windows Server, and much more.

TechNet
MAGAZINE
The Microsoft Journal For IT Professionals

Request your FREE issue today!

http://technetmagsubs.com/?k=DDJJAN

(continued from page S12)
dictionary of name value pairs. The property is *FullPath*.

The project's *csc* task needs to be contained within a project and target. The project provides a scope so that the master-build file can provide settings to the project. The target is a subroutine that the master-build calls to build the project. Both elements require a name attribute. The project requires a default target that it calls when it runs. Within the project's target, I add a property for the target to hold the output assembly name.

I add the *depends* attribute to the target to control the order in which the projects get built. To create the list of target dependencies, the *GetProjectDependencies* iterates over the project's references collection. Each reference that refers to a Visual Studio project gets added to the list of dependencies. The list is a comma-delimited string. When NANT builds the solution, it uses the dependencies list to determine the order in which it builds the solution.

VisualStudio's build process keeps the output in the project's directory, too. The output gets put in a subdirectory of the project's bin directory. The subdirectory is named for the build configuration. Normally, it is either "Debug" or "Release." To implement the build, I use a pair of NANT properties. The first property is the name of the output being built that contains another NANT property that determines the subdirectory to put the output in. The output assembly property looks like Listing Four.

The *outputSubDirectory* property is set in the master-build file prior to calling the project's build. Within the macro project, I wrote two subroutines to make this part of the process easier. The first subrouting writes NANT properties, while the second generates the output path (see Listing Five).

While the *csc* task has lots of attributes that can be supplied, I only use three:

- The *output* attribute is the name of the output assembly; the *outputAssembly* property holds the assembly's name.
- The *target* attribute controls whether *csc* generates a DLL, EXE (console application), or WINEXE.
- The *debug* attribute switches between a debug and release versions for the

A Brief Description Of NANT

NANT is a free .NET build tool (http://sourceforge.net/projects/nant/) that is like MAKE, only better. NANT's build files are written in XML and the tasks that make up the build are implemented using objects written in .NET. The combination of XML and objects make NANT an extremely flexible and effective utility for automating build processes. It is flexible enough to be used for more than just compiling and building products.

NANT is not difficult to use, but getting started can be challenging. My goal is to briefly describe how to use NANT. To build a C# project using NANT, I will use four NANT elements. Three of the elements—project, property, and target—are NANT constructs; the fourth is the CSC task that compiles C# code.

The project element is required by NANT to contain everything else. The project can contain targets, properties, and tasks. Projects can have three attributes—*name*, *default*, and *basedir*. Only the default attribute is required. The default attribute is the name of the first target that gets called. When NANT runs the build script, it executes all of the tasks and properties outside of targets before calling the default target.

The property element is how you declare variables in NANT. Properties can be used in attributes in a build script. This is done by placing the property name inside of "${" and "}". For instance, if you have a property called *foo* you use it like: *${foo}*.

Targets are NANT's subroutines. You place most of your tasks within a target to organize your build script. Each target requires a name attribute. Targets can also have dependencies that cause the dependent target to be executed before the target executes.

The *csc* task compiles C# code using the csc compiler supplied with the .NET SDK. The *csc* task has several attributes, but the main ones to know are *target*, *debug*, and *output*. Targets determines whether the compiler produces an EXE or DLL. If the *debug* attribute is *true*, the compiler produces debug symbols. The *output* attribute supplies the name that the assembly produces. The *csc* task also needs references, sources, and optional resources. These are supplied by sub-elements that go inside the *csc* task.

To compile a C# program, the compiler must be told what other assemblies the program uses. This is done using the references tag. Each referenced DLL is added individually by putting its name inside an *include* element. For instance:

```
<references>
  <includes name="mydll.dll" />
</references>
```

You must provide the complete path to the assembly being referenced or the compiler can't find it.

Source files and resources are handled using filesets—NANT objects that represent a set of files. The set of files can be made by either explicitly adding files to the set or using wildcards. The fileset has a *basedir* attribute that controls the base directory of all the files paths are taken from. To add all of the C# files in the "C:\MyProgram" directory to your sources element, you do the following:

```
<sources basedir="C:\MyProgram">
  <includes name="*.cs" />
</sources>
```

If *MyProgram* has a directory structure underneath it that needs to be included, you can use the following:

```
<sources basedir="C:\MyProgram">
  <includes name="**/*.cs" />
</sources>
```

Normally, wildcards do not match or chase down directories. NANT added the ** operator for that purpose. This makes it much easier to deal with complex directory structures.

Listing One contains the entire build script. The build script has a project whose default target is *myprogram.build*. The project has a single property *debug*. The value of *debug* is set to *true*. The *myprogram.build* target contains the *csc* task, which compiles the C# program and names the output "myprogram.exe."

To run this build, save the NANT script in a file called "myprogram.build." Then start Visual Studio's command-line shell. Add the path to NANT's bin directory to your environment. Change to the project directory and run NANT, which automatically searches a file that ends with ".build" and run its default target. You can also specify a build file by adding the */f* parameter.

Currently, NANT provides 58 tasks in its main distribution and 42 tasks in the NANTContrib package. You can also write your own tasks for anything not already implemented.

—J.W.

ANNOUNCING:

OSDL™ Enterprise Linux® Summit

COME TO THE SOURCE™

January 31 – February 2, 2005 • Hyatt Hotel • Burlingame, CA

Explore Linux Deployment Strategies for Your Enterprise

Summit Highlights:

★ **An "Open Conversation" with Linus Torvalds, Andrew Morton & Brian Behlendorf.** Moderated by OSDL CEO Stuart Cohen, this unique panel of Linux and OSS notables will focus on the increasing importance of Linux and open source software in today's complex IT environment. This keynote is sure to be both informative and provocative.

★ **Utility Computing and the ISV Opportunity on Linux.** Utility computing is approaching a tipping point in enterprise adoption. There are enormous business opportunities for ISVs who can move quickly and capitalize on Linux's early role as the critical platform for utility computing.

★ **Tutorial Sessions** providing valuable, "how to" guidance on Linux architecture, integration, legal, and applications development issues.

★ **Focus Sessions** on critical enterprise Linux topics. Presenters include Andrew Morton, Larry Rosen and Brian Behlendorf.

★ **Networking Opportunities** with peers, Linux leaders, and key vendors.

AND MUCH MORE!

Who Should Attend

Linux system administrators and architects, corporate developers, ISVs and vendors who want to explore the practical solutions for accelerating Linux deployment and increasing Linux business across the enterprise.

THE Source of Linux Deployment Information

Sponsored by OSDL and its member companies

FOR MORE INFORMATION OR TO REGISTER:

ONLINE
www.OSDLLinuxSummit.org/ddj

PHONE
+ 781-444-0404

EMAIL
info@OSDLLinuxSummit.org

(continued from page S14)

output. The *debug* attribute is set to either "true" or "false" by the master-build script. To get the target type, the *GetProjectType* subroutine returns a string that tells the compile whether it is generating a DLL, EXE, or WINEXE; see Listing Six.

Within the *csc* task, you need to supply the references, resources, and source files for the compiler to use. The *WriteReferenceElement* subroutine writes the references for the project. The project object contains a collection of references for the project. The macro needs to handle DLL references and project references. In both cases the result is a path to a DLL. References to DLLs (assemblies) are easily handled. Project references require an extra step. You have to determine the output path first. To get the project output path, build a string using the project's properties for "FullPath" and "OutputFileName." The *GetReferenceProjectOutput* handles building the string; see Listing Seven.

If there are resources in the project, the *WriteResourcesElement* subroutine gets called. If the item in the *ProjectItems* collection is a resource file, it is added to the *resources* element. The actual source-code files are added to the *sources* element by the *WriteSourcesElement* subroutine. Just like the *WriteResourcesElement* subroutine, it iterates over the *ProjectItems* collection. The difference here is that it looks for C# source files; see Listing Eight.

After you have generated a build file for each project in the solution, you can generate a master-build file. Generally, I don't want to automatically generate the master-build script because there are many other tasks that need to occur during a build besides compiling projects. For instance, before compiling the projects, you may want to grab the latest source code from a source repository such as SourceSafe. You may also wish to run unit tests and generate an MSI after you have successfully compiled your projects. Some of these tasks could be generated by a macro.

However, you may inherit a project that does not have an automated build. In this case, having a skeleton master-build script generated would be nice. The master-build script will include all of the project-build scripts and call each project's build once for a debug build and once for a release build. In the process of doing these tasks, it sets the properties for output directory and debug mode. The macro for doing this is in *GenerateNANTSolutionScript*. To make the script easier to maintain and edit, the build is broken into four parts. The *buildAll* target calls the *buildDebug* and *buildRelease* targets. The *buildDebug* and *buildRelease* targets both set up the *debugMode* and *outputSubDirectory* properties, then call the *build* target, which calls the actual projects so they can compile the sources. When the macro is run, it generates the project-build scripts, then generates the master-build script. After you have generated your initial build scripts you only have to regenerate the project-build scripts to maintain the build.

Using the generation of NANT build script, I've demonstrated how to use Visual Studio's Solution and Project objects to make build management easier. Visual Studio provides an object model that is accessible and that lets you extend and expand its capabilities. Effectively used, you can save yourself a lot of work.

DDJ

Listing One

```
Dim projectCollection as EnvDTE.Projects
Dim counter as integer
projectCollection = DTE.Solution.Projects
For counter = 1 to projectCollection.count
    Dim aProject as Project
    aProject = projectCollection.Item(counter)
    if aProject.Kind = prjKind.prjKindCSharpProject then
        GenerateCSharpProjectBuild(aProject)
    end if
next
```

Listing Two

```
<csc ...>
    <references>
        ....
    </references>
    <resources>
        ...
    </resources>
    <sources>
        ...
    </sources>
</csc>
```

Listing Three

```
Dim outputFileName As String
outputFileName = theProject.Properties.Item("FullPath").Value & "\" &
                                                    theProject.Name & ".build"
Dim xmlWriter As New XmlTextWriter(outputFileName, _
                                    System.Text.ASCIIEncoding.ASCII)
xmlWriter.Formatting = Formatting.Indented
```

Listing Four

```
<property name="outputAssembly"
value="C:\work\keep\sample\${outputSubDirectory}\sample.exe"/>
```

Listing Five

```
'Write a NANT property element
Private Sub WritePropertyElement(ByVal writer As XmlTextWriter,
        ByVal propertyName As String, ByVal propertyValue As String)
    writer.WriteStartElement("property")

    writer.WriteAttributeString("name", propertyName)
    writer.WriteAttributeString("value", propertyValue)
    writer.WriteEndElement()
End Sub
```

```
Private Function GetOutputPath(ByVal theProject As Project)
    GetOutputPath = theProject.Properties.Item("FullPath").Value &
                "${outputSubDirectory}\" &
                theProject.Properties.Item("OutputFileName").Value
End Function
```

Listing Six

```
'Determine the project's type (exe, dll, etc)
Private Function GetProjectType(ByVal theProject As Project) As String
    Select Case CType(theProject.Properties.Item("OutputType").Value, Integer)
        Case 1
            GetProjectType = "exe"
        Case 2
            GetProjectType = "library"
        Case 0
            GetProjectType = "winexe"
        Case 4
            GetProjectType = "module"
    End Select
End Function
```

Listing Seven

```
'Returns the path to the references project's output
Private Function GetReferencedProjectOutput(ByVal refProject As Project) As
String
  GetReferencedProjectOutput = refProject.Properties.Item("FullPath").Value &
          "${outputSubDirectory}\" & _
          refProject.Properties.Item("OutputFileName").Value
End Function
```

Listing Eight

```
'Write the sources needed to compile the project
Private Sub WriteSourcesElement(ByVal theProject As Project, ByVal writer As
XmlTextWriter)
    Dim counter As Integer
    writer.WriteStartElement("sources")
    writer.WriteAttributeString("basedir",
theProject.Properties.Item("FullPath").Value)
    counter = 1
    For counter = 1 To theProject.ProjectItems.Count
        Dim projectFile As ProjectItem
        Dim filePath As String
        projectFile = theProject.ProjectItems.Item(counter)
        If "cs" = Right(projectFile.Name, 2) Then
            WriteIncludeElement(projectFile.Name, writer)
        End If
    Next
    writer.WriteEndElement()
End Sub
```

DDJ

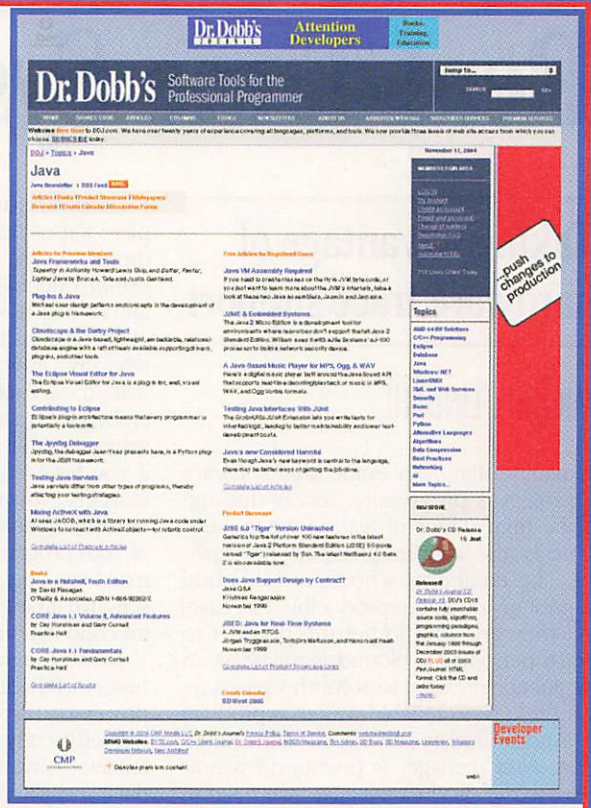
ESSENTIAL INFORMATION FOR SERIOUS DEVELOPERS

On the Web at www.ddj.com

Dr. Dobb's Topic Areas

The premier location for comprehensive technical information for every programmer's needs

The editors of *DDJ* have created these information-packed areas to showcase articles, reviews, FAQs, white papers, and more. Covering a variety of topics, the Topic Areas span the depth of software development from algorithms to real-time computing.



DATABASE DEVELOPMENT TOPIC AREA...

Few applications are more fundamental to business than databases. This topic area keeps developers up to date on the latest in database technology, from embedded to distributed systems. The Database Development Topic Area contains in-depth articles, white papers, FAQs, a discussion forum, related Web links and more!

www.ddj.com/topics/database/

JAVA TOPIC AREA...

This topic area is focused on providing developers the latest technical information and news related to Java. This one-stop shop for complete Java information contains articles, white papers, product information, and events all about Java!

www.ddj.com/topics/java

ADDITIONAL CHANNELS...

Linux / Algorithms / C/C++ Programming / Scientific Computing / Computer Security / Scripting and Alternative Languages / Patterns and Object-Oriented Design / Graphics Programming / Communications and Networking / Real-Time Computing / AI / Data Compression / UNIX / XML and Web Services / Python-Url / TLC-URL / Benchmarking and Software Testing / Basic / Perl / Distributed Computing / Eclipse / Windows/.NET

CMP
United Business Media

VISIT TODAY!
www.ddj.com

Dr. Dobb's JOURNAL
SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

Using Trace to Debug Real-Time Systems

Taking advantage of hardware trace support

MICHAEL LINDAHL

Imagine that your embedded software project is progressing nicely through the final stages of testing when, suddenly, the software unexpectedly crashes. The engineers who found this bug cannot give you a reproducible test case, yet they are able to get the product to crash periodically. Sound familiar? As an engineer, you want to solve this problem, yet there is no reliable way to reproduce it. And to add to the stress of the situation, your manager is pressuring you to release the product because you need to get to market as quickly as possible. Software teams encounter this situation all the time and are forced to release products even though they know that there are serious bugs lurking in the code.

Now imagine that there is a tool that gives you nearly perfect visibility into what your software was doing when the system crashed, making it easy to find and fix these intermittent and difficult bugs. The technology that makes this dream a reality is trace data generated by embedded microprocessors, and the tools that collect and analyze this trace data make it possible for you to analyze what went wrong and find the cause of the undesirable behavior.

Hardware trace data has been around for years, yet it has traditionally only been used by hardware and low-level firmware engineers to track down board-level problems and initialization failures. However, the concept behind hardware trace data is familiar to software engineers because they have been using a similar technique—*printf()* logging and debugging—since the advent of modern software development.

Michael is a software engineer at Green Hills Software. Michael can be reached at mlindahl@ghs.com.

Hardware trace data provides a complete log of the instructions executed by the processor, essentially giving you full *printf()* debugging in a completely nonintrusive fashion. This lets you easily look at interactions between multiple tasks and interrupts in your system. However, hardware trace data can also be used in many embedded systems where normal code instrumentation is less than ideal. For instance, if you have a system that has tight timing margins, you may not be able to insert enough extra statements to track the flow of your program reliably. Additionally, you may have bugs that occur only because of sensitive timing problems and inserting extra code may make the bugs disappear.

Because hardware trace provides a full history of the instructions executed by the processor, there is too much data to analyze efficiently by hand, but there are extensive software tools that analyze trace data and provide high-level information to help solve a variety of problems. Basic trace-analysis software lets you reconstruct the flow of your program and possibly see the memory locations that were read or written by it. There is also software that can generate profiling information from trace data, provide additional high-level views of your program execution, and even software that lets you virtually step backwards through your code so that you can see exactly what happened. These tools offer the main value of trace data because you are able to collect and analyze all of this information without having to change any of the runtime characteristics of your embedded software project.

Hardware Versus Software Trace

Much has been written about software trace and instrumenting code to provide software trace logs (see, for example, "Tracing Program Execution & NUnit" by Paul Kimmel, *DDJ*, August 2004). While this type of instrumentation is great for certain applications, hardware trace has several advantages over software trace, especially for embedded systems.

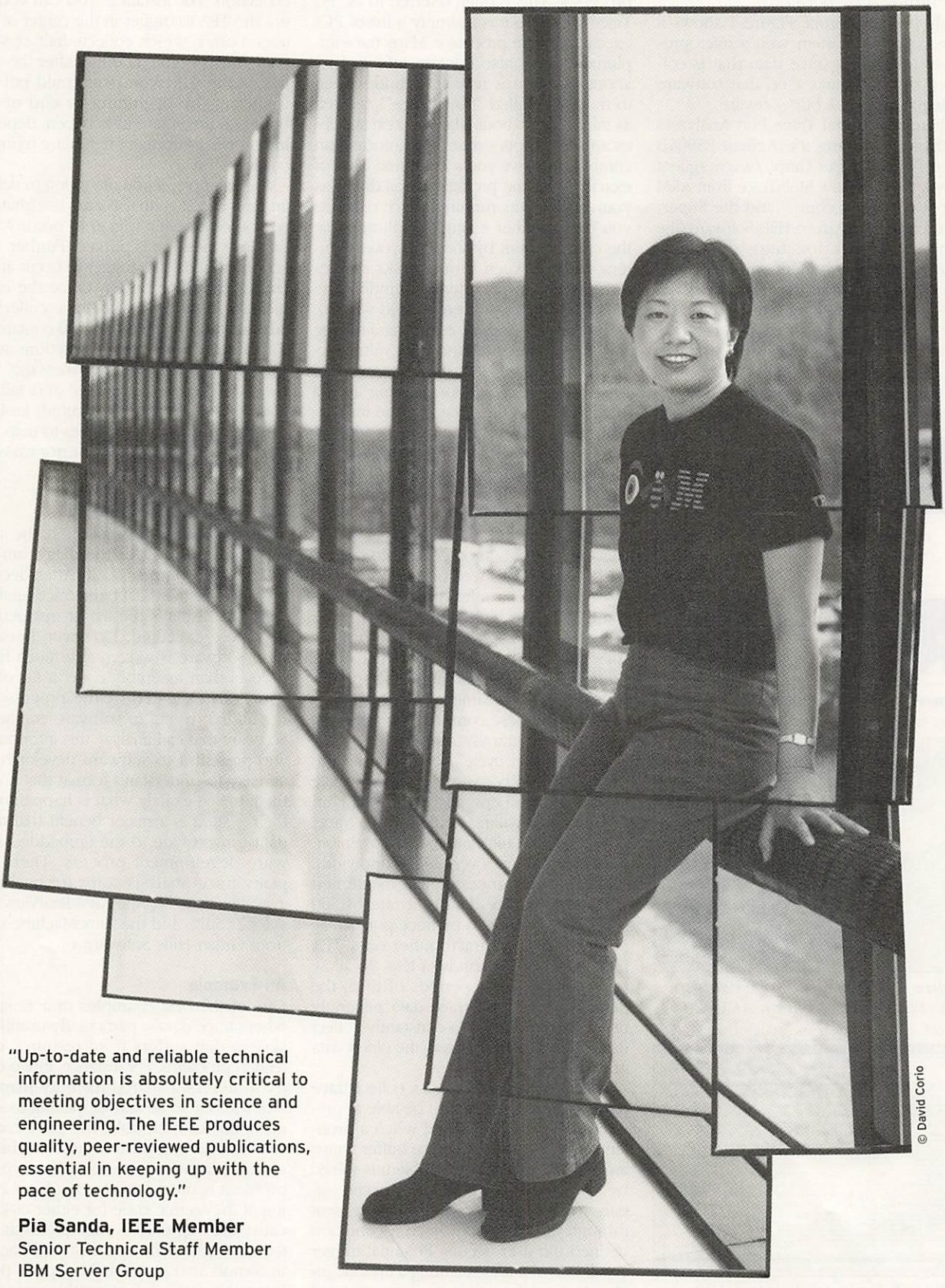
First of all, software trace requires you to instrument your code. If you have an existing codebase that does not have trace instrumentation, it could take months to insert trace statements in all of the interesting code fragments. And if you overlook some of the interesting code or neglect to instrument some functions, you may spend a good deal of time investigating the wrong cause of your bug.

A second advantage of hardware trace data is that it often has the ability to provide data values along with each instruction that is executed. This lets you see the values that caused the program to take the incorrect execution path, which gives you more insight into what caused the problem and how to fix it. While you can certainly log values with software trace, it is impractical to log more than a few, and if a bug occurs and you did not instrument one necessary value then you may not be able to find the cause of your bug.

Finally, the most important advantage that hardware trace data has over instrumentation—especially for embedded systems—is that it does not involve changing any aspect of your program to collect the data. This is vital for applications that may involve timing errors, interrupt latency issues, or other bugs that only appear when a specific set of circumstances arises. Additionally, it is often impractical to instrument code in interrupt handlers or other timing-sensitive code. In nearly all high-reliability applications, you must certify and test the final code without any instrumentation, and software trace almost certainly cannot be enabled in production code, making it impossible to use software trace to track down bugs that appear in the production version of the code.

What is Hardware Trace Data?

Hardware trace data is generated and output by microprocessors and collected using a Trace Port Analyzer (TPA), which is a device that records trace data from a microprocessor and stores it into a trace buffer on the device. This data is output over a dedicated trace port and consists of a stream of highly compressed packets. Some examples of embedded microprocessors that currently support trace are ARM processors such as the ARM7, ARM9, and ARM11; the IBM PowerPC 4xx series; as well as the PowerPC 5554 and MAC7100



"Up-to-date and reliable technical information is absolutely critical to meeting objectives in science and engineering. The IEEE produces quality, peer-reviewed publications, essential in keeping up with the pace of technology."

Pia Sanda, IEEE Member
Senior Technical Staff Member
IBM Server Group

© David Corio

IEEE. Providing the information you need. Join us.
www.ieee.org/member2

IEEE

(continued from page 66)
family from Motorola. Figure 1 shows a trace collection system with a microprocessor outputting trace data that is collected by a TPA and a PC that software engineers use to debug software.

There are several Trace Port Analyzers available, including the Agilent E5904B Trace Port Analyzer (http://www.agilent.com/), the RealView MultiTrace from ARM (http://www.arm.com/), and the SuperTrace Probe from Green Hills Software (the company I work for, http://www.ghs.com/). The trace buffers on TPAs range from a few megabytes (the Agilent E5904B has 4 MB of trace storage) to 1 GB (available on the SuperTrace Probe). Figure 2 shows a SuperTrace Probe ready to collect trace from an ARM920 target.

Trace data essentially consists of a log of each instruction that is executed by the processor. This is often referred to as "PC Trace" because it is a simply a list of PCs executed by the processor. Many trace implementations also contain information about data that is read from and written to memory (called "Data Trace"), as well as information about when context switches and interrupts occur. This information combines to give you a complete view of exactly what the processor was doing as your system was running. Trace data lets you debug either a single application or the entire system by showing you the interactions between various tasks and interrupts. This is essential for finding the most difficult bugs in complicated systems.

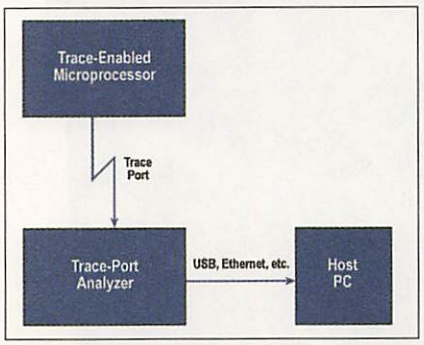


Figure 1: *Hardware trace collection overview.*

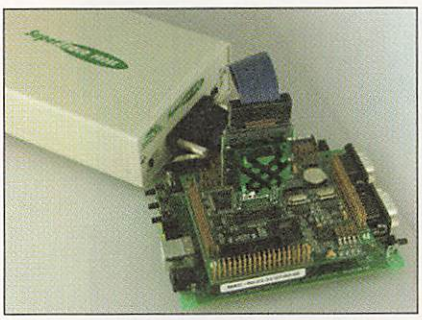


Figure 2: *SuperTrace Probe ready to collect trace data from an ARM920 target.*

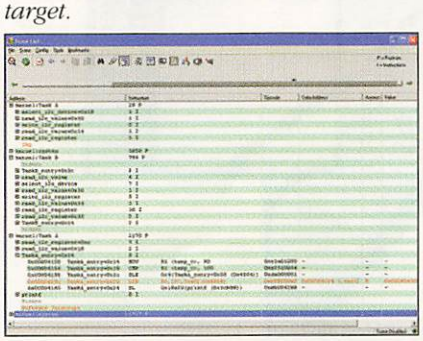


Figure 3: *Trace-analysis tool. You can see that the first access to the I^2C bus by task A was interrupted by task B, causing a resource contention problem.*

As you can imagine, collecting a full log of the PCs executed and data values accessed involves a lot of data. For instance, if you have a processor running at 200 MHz and executing roughly one instruction per cycle, storing a full trace log of 32-bit PCs involves storing roughly 800 MB of trace data per second. On top of this, if 20 percent of the instructions are data accesses (assuming data trace consists of 32-bits of address and 32-bits of data), then that is another 320 MB of data per second. That's well over 1 GB of data per second for a relatively slow processor! Due to this high volume of data, the designers of trace-enabled microprocessors have implemented efficient compression systems to minimize the amount of data that is streaming out of the processor. With these compression schemes, trace ports generally output between 4 and 20 bits per cycle, depending on the compression scheme, whether data trace is available, and how many pins the chip designers are willing to dedicate to trace.

Even with a high rate of compression, trace data still uses very high bandwidth. The 200-MHz processor just mentioned still outputs trace data at a rate of 500 MB/second with a 20-bit trace port. Thus, even a very large trace buffer on a TPA fills up in a few seconds or less. To allow tracing over longer periods of time, the devices that collect trace data generally use a circular buffer to constantly collect data, always throwing away the oldest data once the buffer is full.

Using a circular buffer to collect trace data makes it important to be able to precisely stop trace collection when a problem occurs because the trace buffer wraps around so quickly. This problem is solved by "trace triggers," which are special signals from the trace hardware that are sent through the trace port when a condition defined the user occurs. Potential trigger conditions include executing a specific instruction and reading from a selected memory location. When the TPA receives the trigger signal, it collects a user-defined portion of the buffer and then stops trace collection. For instance, you can configure the TPA to trigger in the center of the trace buffer, which collects half of your trace buffer before and half after the trigger occurs. Likewise, you could put the trigger at the beginning or end of the buffer, or anywhere in between, depending on what information you are trying to collect.

For example, if you are trying to debug an error message that occurs intermittently, you could set a trigger to occur when the error message is printed. Further, you could configure the trigger to occur at the end of the trace buffer. Once the error message is printed with trace collection enabled, your trace buffer will contain information about what instructions were executed prior to the error message. You will now be able to examine in detail the code paths that were executed, looking at memory and register values to help you determine what caused the error message you are interested in.

Trace-Analysis Software

Once the trace data is collected, the analysis software uploads the data from the TPA and decompresses it. After decompressing the trace data, the trace-analysis software displays the list of instructions that were executed. However, because most software is written at a much higher level than assembly code, a list of instructions executed by the processor is not all that useful to software engineers. Software that can analyze this information and present it to software developers in an easy-to-understand format that allows them to see exactly what is happening in their system is a major benefit that trace data can provide to the embedded software development process. There are many trace-analysis software packages available, including the ARM RealView Developer Suite and the TimeMachine tools from Green Hills Software.

An Example

One of the best examples of a situation where trace data is particularly useful is a system that suffers from resource contention problems. For instance, if two tasks try to access a shared memory-mapped bus without some sort of locking mechanism, one or both tasks will get incorrect values. This could cause any number of system glitches and errors. This type of problem is often not obvious from looking at the source code for either task because the bug only occurs due to the interaction between two tasks. Additionally, in complicated systems, this type of problem may occur infrequently, making it even harder to track down the root cause. With trace, as long as you catch the bug occurring one time, you have a good

chance at discovering the root cause and fixing the bug.

Listing One is a simplified example of two tasks incorrectly sharing a resource. In this case, two tasks read from the I²C bus on a processor. Roughly, an I²C bus is a serial bus requiring multiple transactions to read a single piece of data, including selecting a device and then reading from the selected device. Task A periodically reads from a temperature sensor on the I²C bus and task B reads from an EEPROM that is also connected to the I²C bus. Task A checks for an invalid reading from the temperature sensor and prints an error when it gets an invalid reading. You can set a trigger to occur when the error condition (*printf()* statement) is executed. Then enable trace when you start running the program and wait for the trigger to be hit. Once the trigger is hit, the trace data will be loaded into the trace-analysis software and you will be able to look at what happened in your system prior to the error.

Many trace-analysis software packages convert the list of instructions executed by the processor into a list of functions that were executed and what task was running at each point in time. This is a relatively elementary type of analysis that can give useful information to programmers debugging a system. For this example, Figure 3 shows a sequence of functions and context switches that indicate that a context switch occurred after selecting one device on the bus but before reading the data from that device.

From examining this series of function calls, it is clear that one task initiated an I²C bus transaction while another bus transaction was in progress. However, especially since these two devices seem unrelated at first glance, traditional debugging techniques will probably not uncover this bug nearly as quickly as the trace tools.

Performance Analysis

Trace data also makes performance analysis easy by processing a trace log and displaying profiling information. This lets you easily see how much processor time each part of your system uses without requiring any code instrumentation or other overhead. Additionally, because it does not require any code modifications, profiling information generated from trace data lets you see how long various interrupt handlers take when they run and let you identify situations where various components will not meet their real-time deadlines. Another significant benefit of using trace data for profiling is that you can get very accurate performance numbers without incurring any system overhead or using on-board sampling timers, which lets you collect all instructions for a region of code rather than some relatively small number of samples per second.

Many TPAs are able to generate either a timestamp or cycle count information along with each incoming trace packet. A timestamp is the time at which each incoming trace packet is collected, often with a resolution of a few nanoseconds, and a cycle count is the number of processor cycles that each instruction takes to execute. Both of these pieces of information provide accurate timing information for individual instructions, memory accesses, and other processor events. You can even see individual cache misses by finding memory accesses that take longer than others! This information provides very detailed performance analysis, letting you see detailed performance information that is unavailable using typical debugging methods.

One great advantage of this profiling information being readily available when tracing a microprocessor is that you can often find problems that you may not even be specifically looking for. Since you do not need to recompile the code to collect profiling data, it is almost effortless to collect trace data and convert it to profiling information. You can then easily see what functions executed and find any potential performance problems before they start negatively affecting your system.

Another specific application of trace-based performance analysis is driver optimization. If you have ever had to try to squeeze every bit of performance out of an Ethernet driver and/or TCP/IP stack, you know that a large part of the battle is determining exactly what sequence of events occur and where the bottlenecks are. Using trace data coupled with the generated profiling information allows you to first find the functions or segments of code that take the most time, then examine the trace data to determine exactly what code was executed and why it took so long. Whether it is because your code copies the data too many times or uses an inefficient algorithm, trace tools can quickly help you find the bottlenecks in your system. You can then examine these areas in more detail using traditional debugging techniques to eliminate inefficiencies in your code.

Conclusion

While trace is currently supported on relatively few embedded processors, the advantages of devoting some portion of the silicon to generating trace data greatly outweigh the extra costs in many applications. Indeed, since more and more new processor designs include trace implementations and trace-analysis tools are improving, trace data has the potential to fundamentally change the way software engineers debug their software. From simple resource management errors to complicated memory bugs within a single application, trace data and analysis make it possible to correct otherwise difficult bugs in a relatively short time.

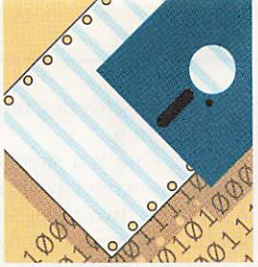
How many of us have had bugs reported to us that we were unable to reproduce and had to write off as random glitches? Imagine if it were suddenly possible to solve many of these problems and remove most if not all of the glitches from your system. With trace data, this possibility is becoming a reality. And as more processors support trace and the trace-analysis tools improve, it will only become easier in the future.

DDJ

Listing One

```
// This function is obviously not correct for actually reading
// from an i2c bus, but is an example of a function that could
// read from a shared resource.
int read_i2c_value(int DEVICE_ID)
{
    // Select the I2C device
    select_i2c_device(DEVICE_ID);
    // Select the register on this device that we want to read
    write_i2c_register();
    // Actually read the register
    return read_i2c_register();
}
#define TEMPERATURE_SENSOR 1
#define EEPROM 2

void TaskA_entry()
{
    while(1) {
            int temp_c;
            ...
            temp_c = read_i2_value(TEMPERATURE_SENSOR);
            if(temp_c > 100) {
                // Error.  The temperature cannot possibly be
                // above boiling!
                printf("Error reading temperature sensor!\n");
            }
            ...
    }
}
void TaskB_entry()
{
    while(1) {
            int eeprom_value;
            ...
            eeprom_value = read_i2c_value(EEPROM);
            ...
    }
}
```

DDJ



Random Thoughts

Michael Swaine

Randomness is a glittering snowflake of a concept: utterly familiar yet beautiful and intricate when examined closely. And when you examine it too closely, it evaporates, leaving you wondering if it was just something you imagined. Science writer Ian Stewart recently posed the question of whether randomness even exists, or if it is just an invention of our superstitious minds ("Randomness," *New Scientist,* September 25, 2004).

Real or imaginary, randomness comes in flavors and displays itself in facets: It can be classical or quantum, a mark of superstition or a measure of ignorance, certain or uncertain, ergodic, or chaotic. It's related to heat, and you can improve algorithms by "heating" them up with some gratuitous randomness; but it's also related to noise, and allows you to sort noises according to their "colors."

Randomize bits and you have a cryptogram. Randomize letters and you have an anagram. Randomize thoughts and you have a Paradigm, or at least a "Paradigms" column. At the end of this month's random walk, I'll derandomize the anagram I published here three months ago.

Superstition

Conventional wisdom (which is to say, random public opinion) holds that there is "real" randomness at the heart of all things, a peculiar quantum randomness that cannot be removed. This, CW says, is different from "classical" randomness, which is just a measure of our own ignorance and can be reduced by acquiring more information.

But if classical randomness is simply a statistical measure of observer error, that's not how we talk about it. We speak of random processes as though the processes in question were not deterministic. When we can't find the pattern behind a sequence of events, we give the unknown pattern a generic name—randomness—rather than admit that the case is not closed. But pretending that you've said something meaningful when you've merely given a name to your ignorance: Isn't that superstition?

Uncertainty

Quantum randomness, though, is "real" randomness. Physicists say that there is no statistical process underlying quantum randomness. It just is. And the case seems solid, offering no loophole. Or at least that the possibilities for getting around Bell's Theorem and the empirical evidence for it are few and unpretty. You have to assume, for example, that certain quantities simply cannot be computed, or that a certain form of argument is inadmissible.

But if you're willing to reject one of the assumptions of Bell's Theorem, you can in fact construct a hidden-variable theory, in which the observed quantum randomness is merely a statistical property of unobserved variables. So the question remains: Is quantum uncertainty certain?

Contrariness

Randomness, or unpredictability, seems to have some affinity to irreversibility. Fill the grid in the game of Life or some other cellular automaton with random bits, and let it evolve through many generations. What you see is "a complex web of structure and activity...with various levels of organization" (Toffoli and Margolus, *Cellular Automata Machines,* MIT Press 1987; ISBN 0262200600).

But if the rule of evolution of the automaton is time reversible, the system looks just as random at every step. Only irreversible rules lead to interesting outcomes. Does this say something interesting about reversibility and the universe in which we live?

Anarchy

Whatever randomness is, it is certainly related in some way to unpredictability. Yet the relationship between randomness and unpredictability is far from straightforward. Randomness can lead to predictability, while deterministic processes can turn out to be unpredictable. In fact, can be expected to, in certain circumstances. It's predictable unpredictability. You doubt? Consider: Statistical mechanics is the mathematical engine of thermodynamics. Boyle's Law is a deterministic thermodynamic law relating the pressure, volume, and temperature of a quantity of a gas. If you know two of these quantities, you can always compute the third. It's the law.

What's going on at the molecular level, though, is as anarchic as a food fight at Animal House: individual gas molecules randomly bouncing off one another. (I don't mean to slander these molecules; their ricocheting properly obeys Fast Eddie Felson's Law of Combinations. But in the sense that you have no idea of the positions and velocities of the particles, their motion is random.) Yet out of this randomness comes the strict orderliness of Boyle's Law.

Chaos

So much for predictability out of randomness.

To get unpredictability out of determinism, we need look no farther than chaos (which, in my office, is never far away). Chaotic motion, Manfred Schroeder of Bell Labs claims, is much more widespread in nature than regular motion

Michael is editor-at-large for DDJ. *He can be contacted at mike@swaine.com.*

(*Fractals, Chaos, and Power Laws,* W.H. Freeman, 1991; ISBN 0716721368). In chaotic motion, small initial errors grow exponentially until they swamp out the effects of regular motion. It was mathematician and meteorologist Edward Lorenz who found the poetic image for this exquisite sensitivity to initial conditions: Does the Flap of a Butterfly's Wings in Brazil set off a Tornado in Texas? Answer: It might. Tweak the initial conditions a tiny bit in a chaotic model of the weather, and you get radically different weather.

So random processes can have effects that are highly predictable, and deterministic processes can have effects that are essentially unpredictable.

Confusion

Still, on the spectrum of exoticness, we think of chaos and unpredictability and randomness as being toward the exciting and mysterious end, and determinism and statistics as being over near the dull and understood end. Chaos theory lends itself to colorful metaphors like "butterfly effect" and to trippy philosophical discussions. Statistics, in contrast, seems like the most settled of disciplines.

However, there are real differences among statisticians regarding the deep meaning of it all. Statistics is, on careful examination, rather mysterious.

The two philosophical paradigms of statistics are classical(?) and Bayesian. The more you examine either of them, the less satisfactory does it seem. (It seems to me.) One is disturbingly circular in its logic, and the other is disturbingly subjective.

In classical statistics, the statement that a 95 percent confidence interval for an unknown parameter runs from -1.5 to +2.4 makes it sound as though the parameter lies in that interval with 95 percent probability. But if you thought that, you would be thinking right-side-up, which will get you nowhere in classical statistics. All you are allowed to conclude is that if you were to carry out similar procedures time after time, then the unknown parameter would lie in the confidence intervals you constructed 95 percent of the time. Whatever good that is. The developers of classical statistics constructed a lot of very handy tools, and told the people using them that they couldn't actually use them to come to any conclusions that a researcher would be interested in reaching.

Bayesian statistics treats probability as ignorance and provides methods for determining how much your ignorance is reduced by successive collections of evidence. But you can't begin to apply Bayesian methods without first taking a leap of faith for whose justification Bayesian statistics is utterly unhelpful. If you asked a Bayesian weatherman whether it was going to rain tomorrow, his first step would be to ask, "What do you think?"

And these unsatisfying options are the only philosophical models for statistics we have to choose from.

The fact that we can't even agree about what we're really doing when we do statistics is further evidence that we don't really don't know enough about ignorance. Should we be concerned? Is what you know that you don't know more or less dangerous than what you don't know that you don't know? We don't know.

> "Hiding the fact that information has been embedded in a digital file is a tricky process"

Turmoil

We do know that injecting a little randomness into a system can sometimes improve its performance. In his 1951 essay, "Can Digital Computers Think?" (cited in *Alan Turing: The Enigma,* Andrew Hodges, Simon and Schuster, 1983; ISBN 0671492071), Alan Turing played around with the idea of granting "free will" to a computer program by injecting it with a little randomness, and even alluded to by-then decades-old arguments that the indeterminism of quantum processes could help to explain human free will. It seems plausible, based on extant evidence that he was very serious about this latter notion (*Computationalism,* by Matthias Schuetz, MIT Press, 2002; ISBN 0262194783).

But adding randomness to produce smarter algorithms is more than a notion. "Simulated annealing" is a proven technique for raising the "temperature" of a search algorithm, thus increasing the likelihood that it will jump out of a locally optimal state and discover more nearly globally optimal states. In an uncertain universe, a little uncertainty in our algorithms proves to be advantageous.

Noise

One system for categorizing different types of randomness is the color system for noise. Noise, or time-based random data, can be characterized as white, brown, pink, or black, at least.

"[T]he power spectra (squared magnitude of the Fourier transform) of statistical time series, often known as noises, seem addicted to simple, homogeneous power laws in the form f^{-b} as functions of frequency f [Schroeder]." For white noise, the exponent b=0. For brown noise, akin to Brownian motion, the exponent is –2. Pink noise, with an exponent of –1, lies between; black noise has an exponent beyond –2. The color expresses how time-dependent the randomness is.

Hodge says that Turing enciphered Winston Churchill's voice into white noise while working on crypto work during WW II, but Hodge isn't necessarily being as precise here as Schroeder. It could have been some other color of noise. In fact, Schroeder points out, white noise is a fiction. Its spectrum can be flat only over some finite frequency range. Whatever the range, it's definitive noise, staticky and unpleasant to the human ear.

Pink and black noise are widespread. Black noise governs power blackouts, floods, droughts, and bear markets. Pink noise has equal power in octave frequency bands, and because of the unbiased way it stimulates the inner ear, it is the psychoacoustic equivalent of white noise. It crops up in many natural systems. Brown noise shows up in computer graphics systems that produce fractal mountain ranges, and in all sorts of other places.

Obfuscation

The signature use of randomness, though, is in cryptography. The voracious appetite of crypto applications for random numbers is rich in irony. Not only are large, inherently uninteresting numbers now a commodity desired for their very uninterestingness, but the use to which these uninteresting numbers are put is to disinform. Crypto celebrates randomness not as in its informing use as a measure of ignorance but in its disinforming use as a means of ensuring ignorance.

But what exactly is a random number? If "random" just means "I don't know where that came from," how do you, as programmer, generate one? We've all written random-number generators, but we all know that the numbers generated were only "random enough" for our purposes. Programs that are said to produce random numbers actually generate pseudorandom numbers unless they are attached to gamma-ray detectors or some other source of measurements of quantum uncertainty.

But that's not quantum crypto. Quantum crypto draws on quantum randomness, but in a more fundamental way. It requires specialized hardware that can store information in the form of qubits in superpositions of quantum states, and specialized algorithms that can take advantage of the quantum storage. Quantum crypto promises to be uncrackable, assuming all those aforementioned Bells' Theorem assumptions of quantum uncertainty. And assuming many much more mundane assumptions regarding the effectiveness of the specific implementation's safeguards against spoofing and other non-quantum tricks.

Frankly, the idea of uncrackable encryption scares me. I don't trust myself that much. I recently turned on Apple's FileVault on-the-fly file encryption, and despite the fact that I have two safeguards for getting into the files if the on-the-fly encryption goes on-the-fritz, I'm still nervous.

Concealment

Another approach to obfuscation is steganography. I like the name and I love the concept—hiding in plain sight. Steganography is an alternative to encryption, and is as old as codes. In ancient Greece, they would etch messages in a block of wood, then fill the grooves with wax. Melt the way and read the message. Or they'd shave a slave's head, tattoo a message on his scalp, and let his hair grow back.

According to Wikipedia, "[s]teganography works by replacing bits of useless or unused data in regular computer files (such as graphics, sound, text, HTML, or even floppy disks) with bits of different, invisible information. This hidden information can be plain text, cipher text, or even images." Although steganography is an alternative to encryption, it can be used in combination with encryption, the encrypted message being further hidden steganographically. Or a separate message can be hidden steganographically in the same file with an encrypted message.

Steganography took a back seat to cryptography for years, but began to get some attention in the '90s, even getting its own international conferences. The application that kicked up the popularity of steganography is watermarking. Hiding a copyright message in a file is a way of protecting it that depends on the existence of the message being undetected. This is the unique feature of steganography; encryption doesn't hide its existence.

Hiding the fact that information has been embedded in a digital file is a tricky process. It's a game, really: You have to guess what statistical properties of the file might be analyzed. One trick is to embed the message not in individual bits but in the parity of a set of bits. That way you can change whichever bit in the set is least informative to potential snoops. Google "steganography" for algorithms and more info.

Troublemaking

All this discussion of cryptography and steganography set me thinking.

My personal views on the Second Amendment are: (1) that the founders were describing a means of safeguarding one's rights and one's person against any and all threats, (2) that the idea that the government might be one such threat was no stranger to their thinking, but (3) that in the 21st century it's more than a little naive to think that you're going to defend yourself against the government of the United States with your home arsenal.

I believe that what the authors of the Second Amendment would be defending today is the right to use crypto. I think that crypto occupies the role that guns did two centuries ago. You may disagree, but you could grant me the point for purposes of argument; I'm going somewhere else with this anyway.

I admit that crypto lacks some of the features of a gun. One important feature of the right to carry a gun is that it is self reinforcing, or self defending: If you are carrying a gun, people are reluctant to challenge your right to carry a gun. They may decide not to raise the issue. Using crypto doesn't have this virtue: It doesn't make you a less appealing target for suppression of your right to use crypto, but a more appealing one.

Steganography, however, does make you less of a target. Hiding that you're hiding, that's the trick to hiding from those who want to deny you the right to hide. Stego plus crypto: The un-gun gun?

Distraction

In my October 2004 column, I asked you to consider what obsolete machinery was suggested by the letters "ISO and the URL," and added, "[a]t the risk of sending you off on the wrong track, I'll suggest that you think about locomotives, focusing on the rail sound in south Ireland."

The solution is a Linotype machine. Along with "ISO and the URL" there are two other anagrams in the clues—"the rail sound" and "south Ireland"—that resolve to the strings "etaoin shrdlu."

These are not words, but they are by some estimates the most common 12 letters in English and, more to the point, they are the first two banks of keys on a Linotype machine.

Error

In that same column, I also wrote: "By this logic, all sorts of portable devices should be banned, including memory sticks. Hard to police this policy...Not a crazy idea, just a hard one to enforce, as are all attempts to bottle up information."

A reader writes: "I wanted to make you aware of Microsoft's efforts to help people police this policy: http://news.zdnet.com/2100-3513_22-5356485.html. Personally, I agree that its futile to try and bottle up information."

DDJ

Dr. Ecco Solution

Solution to "Efficient Purity," *DDJ*, December 2004.

To get a solution that has 10 grams of *A* and is 75 percent *A*, follow the design of Figure 1.

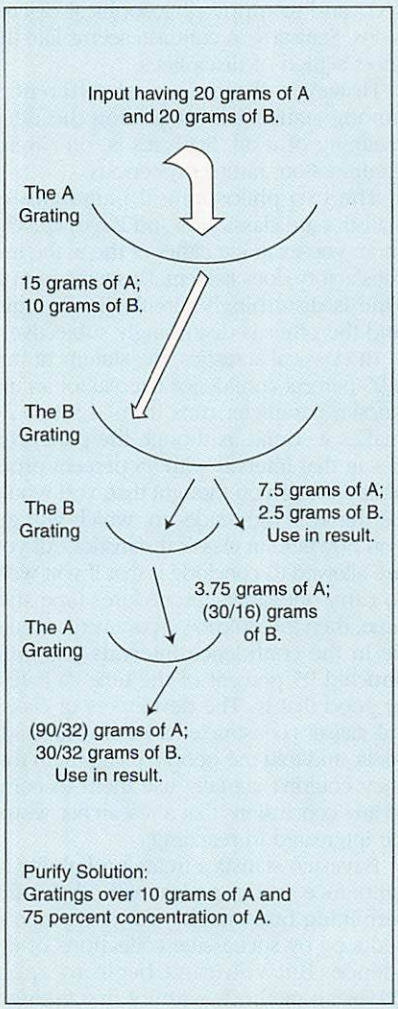
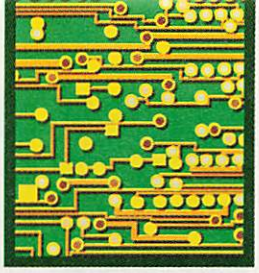

Input having 20 grams of A and 20 grams of B.
The A Grating
15 grams of A; 10 grams of B.
The B Grating
The B Grating
7.5 grams of A; 2.5 grams of B. Use in result.
3.75 grams of A; (30/16) grams of B.
The A Grating
(90/32) grams of A; 30/32 grams of B. Use in result.
Purify Solution: Gratings over 10 grams of A and 75 percent concentration of A.

Figure 1.



ESC Boston

Ed Nisley

The day after I returned from last September's Boston Embedded Systems Conference, my friend Ess mentioned an interesting conversation he had while snaking a hands-free cellular-phone wire through his car's dashboard.

A well-dressed guy walked by, saw what Ess was doing, and they got to talking about those little touches you add to a favorite car. Ess, who's something of a tinkerer, showed off several mods, including his ExxonMobil SpeedPass epoxied onto the gas cap.

The guy observed that most people carry their SpeedPass on their key ring. It turned out he'd done a stint of valet parking and, whenever he and his compadres got a keyring with a SpeedPass, they'd use pliers to extricate the RFID chip, reseal the case with a dab of glue, and enjoy free gas for a month.

As they say at http://www.speedpass.com/: "Always take your Speedpass device with you when you are away from your car, or when you leave your car keys with someone else—for example, in situations when you drop off your car for valet parking or for car repair service."

What's wrong with this picture?

This year, Embedded Security Seminar sessions ran in parallel with the familiar Embedded Systems Conference, describing the state of the art and where we must go from here. An emphasis on security is a welcome sight for an industry that's traditionally produced user-hostile interfaces with no security whatsoever.

Now that we're starting to build friendly equipment, though, we must contend with definitely hostile users, if not overt attackers. Worse, those attackers will have the hardware in their hands, because that's what ubiquitous computing really means.

Who Do You Trust?

In the human realm, you must earn the trust of others, not assume it. You cannot, for example, expect them to simply hand you their money based on your assurances that you are an honorable person. Conversely, you will give someone else your money or your information only after you have established that they will handle it appropriately.

Trust can be personal, based on a long-standing relationship, or institutional, based on the rule of law applied to business transactions. While you may not know that big-box retail clerk personally, you can safely assume she will not make a bootleg copy of your credit card and that, if she does, the law is on your side.

In the network realm, however, trust is based entirely on the blind faith that a program professing to be trustworthy really is. Do you know exactly what an ActiveX control means when it claims to have a valid signature? Do you accept an RPM based on a valid MD5 checksum? Have you ever installed an expired or invalid security certificate for a "known-good" web site?

Embedded devices must handle the mechanics of security without human intervention because, as more and more devices sprout network connections, fewer and fewer humans can oversee their action. Even if you wanted to keep track of every electrical outlet and switch in your house, every can in your pantry, and every article of clothing in your closet, how often would you be willing to run a security audit or even check the logs?

Not very often, if our experience with home computers is any indication.

Scaling Factors

Getting security right for a single, PC-oid network node may be a doable, albeit difficult, project, but creating a solution that scales with the massive deployment of minuscule gadgets is a whole 'nother can of worms. Kevin Ashton's keynote speech on securing the "Internet of Things" showed projected annual shipments of 500×10^9 RFID tags in the U.S. by 2010. That means RFID tags will chew through nearly 40 bits of address space annually.

He pointed out that Wal-Mart currently sells 60×10^9 items per year. If each item carries a tag with a unique address, which is the way RFID tags work, then another factor of 10 for the whole industry over half a decade isn't much of a stretch. The fact that tags are currently too expensive by two or three orders of magnitude remains a problem, even if it's just a Simple Matter of Engineering.

Current RFID tags have essentially no built-in intelligence, as they simply transmit their fixed serial numbers in response to an interrogation. There's a strong desire for tags with writable information, tags with analog sensor inputs, and tags that can support mesh networking. The real problem is power consumption, which rises with each additional function in a device with no room or budget for a battery.

Ed's an EE, PE, and author in Poughkeepsie, NY. Contact him at ed.nisley@ieee.org with "Dr Dobbs" in the subject to avoid spam filters.

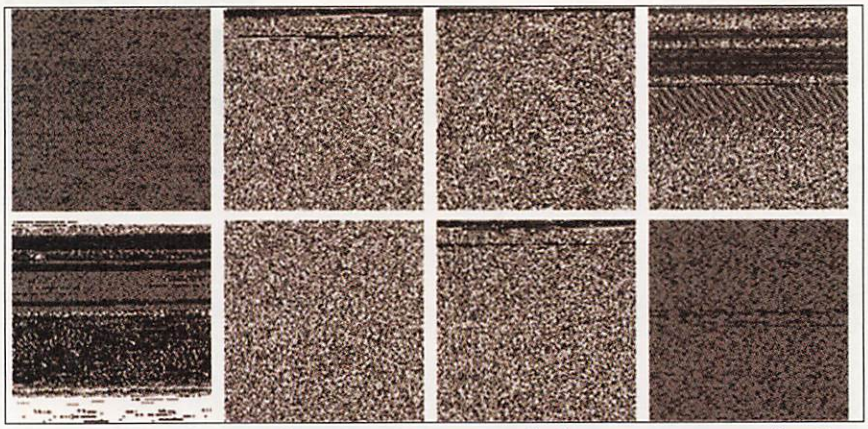
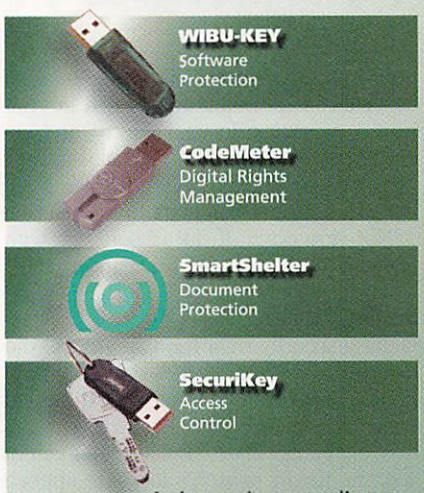
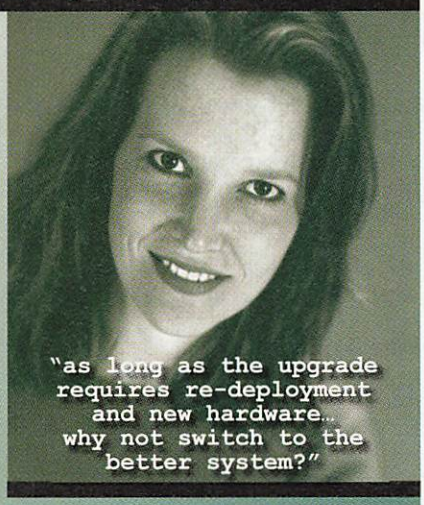


Figure 1: *Converting raw bytes into images reveals the inner structure of each file. You can see that trying to hide a high-entropy key on disk won't work. Left-to-right, from the top: html, jpg, mp3, pdf, ppt, random, sxw, and txt.*

SUPERIOR DIGITAL RIGHTS MANAGEMENT SYSTEMS





Are you facing major re-tooling or re-implementation... all in an effort to keep up with the upgrade? All new hardware to ship? New configurations to deploy? Why put yourself, your staff, and your customers through that and still end up with something less than the best? That same re-tooling can earn you more effective protection that is much more cost effective.

Superior Protection

Hardware and software combined present the most secure protection. But just because there is a hardware key, does not mean you are secure. Challenge-response schemes can be compromised. WIBU System's RID/RED and AXAN technologies provide unparalleled peace of mind. WIBU Systems has consistently and successfully challenged the hacker community. No other protection provider can make that claim.

Economical Solutions

You likely know that the per-unit cost of the hardware is only a small part of the equation. Implementation is at the heart of keeping your costs manageable. The WIBU system provides you with the tools that make multi-platform and multi-port development quick and easy. And, if your product is available as a network license... there is no need to switch hardware. WIBU-KEY excels in both single and multi user environments.

WIBU SYSTEMS

more effective... more efficient... it's just Smart Business!

Order your test kit now!

www.WIBU.com www.GRIFTECH.com/ddj 800-986-6578

He also pointed out a problem that may not have a good technical solution. Suppose everything has a unique ID tag, whether networked or not. Now, what happens when a can of soda, last tracked to your hand, is found at a crime scene? How will your testimony stack up against automatic records that, presumably, can't be faked?

Keeping Secrets

Several talks involved Atmel's AT97SC3201 Trusted Platform Module (TPM), an implementation of the Trusted Computing Group's spec for an on-board secure device, and how to build trusted embedded applications. The TPM is a single-chip microcontroller with crypto accelerator hardware, comprehensive physical tamper-proofing, well-verified firmware, and a straightforward system interface. It handles all the hocus-pocus required for public-key crypto, SHA hashing, digital signing, and so forth and so on.

Systems without a TPM must store their crypto code and keys on a hard drive, Flash-ROM drive, or a similar device, making them vulnerable to snooping and alteration. While you might not make the naïve mistake of putting your private key in a file called, oh, MyPrivate.Key, that's not the only security risk.

One slide in Steve Kinney's talk demonstrated how crypto keys stand out from the usual clutter. Figure 1 is my version of his slide, displaying the first 128×128 bytes of several files on my system as grayscale images. Pure random data is conspicuous because it has very high entropy: Uncorrelated values and no large-scale patterns. Although compressed files, such as JPG and MP3 data, also have high entropy, they also tend to have obvious headers and structures.

Although the TPM contains all your private keys, your system must authenticate itself before the TPM will begin operation. This requires, naturally enough, Yet Another Key stored outside the TPM, which means that an attacker with physical access to the hardware can compromise the system.

The attacker can monitor the data passing between the system and the TPM, locate the external key, help themselves to the data, and break the security. What the TPM can do is encapsulate the crypto machinery and prevent a wide variety of software attacks on your code and data, so it's very good at what it does.

Bill Gatliff moderated a Shop Talk session, much too early on Tuesday morning, on "Experiences in Secure Hardware Design." He was contracted to cook up embedded Linux for a hardware gadget to be sold in a country with, shall we say, a relaxed attitude toward Intellectual

Property ownership. The desired result was a combined hardware/software system that couldn't be duplicated, even after the new owners pried it apart for study.

The bottom line turns out to be that you can't protect your IP in such a situation. The "attackers" are at least as smart as you are, they have access to at least the same level of tools, they have far more time on their hands, and they have all the incentive you can imagine.

Participants in several sessions described various means of tamper-proofing, tamper-resisting, or at least making tampering more obvious. The usual panel switches are completely passé: Expect a pressure sensor in a vacuum-sealed box, an optical sensor watching a particular reflection from a hologram grating glued to the lid, or minuscule wires epoxy-bonded across assembly seams. The folks who build gambling machines have some hairy war stories, too.

Regardless of those precautions, an attacker can simply buy (or steal) enough units to figure out how you've done the security, then defeat it with ease. In this Internet Age, one break is all it takes. Remember, they're at least as smart as you are.

Verily: To own is to be root.

Moving the World

Perhaps the single distinguishing feature of embedded programming, in contrast to ordinary applications, is its intimate relation with the real world. Embedded systems typically react to physical stimuli and produce mechanical outputs, rather than rely on I/O filtered through layers of abstraction. Timing is, of course, everything in that realm.

Vince Socci, from On Target Technology (http://www.ontargettechnology.com/), discussed "Actuator Control Design for Safety-Critical Applications" in the worst-possible time slot—Thursday afternoon. Surprisingly, his talk was well-attended and downright fascinating, at least if you like this sort of stuff.

He pointed out the myriad ways that components can fail and how difficult it can be to ensure that those failures do not cause an overall system failure. Simply adding more parts does not necessarily improve the system's reliability, particularly if reliability isn't at the top of the requirements checklist.

A system can't be just a little bit redundant: Either you ensure that no single failure can bring it down or you don't. You must take probabilities into account, as not everything can be redundant, but you must also work with Murphy's Law in mind.

He gave a particularly interesting example of a transmission for a (very) high-end automobile that incorporated two clutches to avoid losing power during a shift (I hate it when that happens. Don't you?). A design review showed that a hydraulic valve failure could activate both clutches and destroy the transmission. Some mechanical redesign averted that, but then they realized how a microcontroller failure could have the same effect. Solution? Duplicate the microcontrollers and add failure-checking circuitry.

"Current RFID tags have essentially no built-in intelligence"

Triple-redundant systems go one step further, but three may not suffice for airplane control surfaces. A quad-redundant control system has four parallel control channels, from input sensors to output mechanics, arranged so that it will continue to operate normally even with one channel actively opposing the other three. If two channels fail, the system enters a fail-passive mode that can use an alternate control path. If three of the four channels fail, the control system enters a fail-safe mode that will, at least, not actively try to kill you.

Even though automotive drive-by-wire systems demand the same reliability as aerospace systems, they have diametrically opposed constraints on cost, development time, and product volume. An automobile's environment can be more hostile than a combat aircraft (that is, apart from weapons fire), so you don't get any relief there, either.

The design process for a safety-critical system begins with a simple premise: If a critical part fails, your customer dies. He points out that various industries have evolved many different ways to focus adequate attention on the design process, so that adopting a particular design methodology and conforming to a given standard may well be mandatory.

Software poses a particular problem for safety-critical products because software failure modes tend to make hash of traditional failure models. A mechanical actuator has a limited number of failure modes: stuck-on, stuck-off, frozen-in-place, and so forth. A single software failure, on the other hand, can produce bizarre results. Consider that an early version of the F-16 flight-control software would have rolled the plane 180 degrees when it crossed the equator, had that error not been caught in simulation.

In short, if you're entering the safety-critical arena, expect that your schedule and costs will increase by perhaps a factor of five. That's what it takes to produce dependable systems, which should be a wake-up call in the face of schedule and cost pressure.

Reentry Checklist

There was some talk of Smart Dust, which involves computing power in pepper-flake packages, and how it can be a Force for Good. Alas, there's no room for Security As We Know It in those gizmos, so there's lots of room for hackers. Spoofing SpeedPass chips and hacking RFID tags will be just the beginning.

Information on the semiannual Embedded Systems Conferences is at http://www.esconline.com/. The Microsoft Windows Embedded Essentials Seminar, scheduled for all day Tuesday in Ballroom A, was abruptly canceled. I haven't been able to find out what happened.

The Trusted Computing Group is at https://www.trustedcomputinggroup.org/home/ (note the https protocol). More on Atmel's Trusted Platform Module is at http://www.atmel.com/products/Embedded/, although the good stuff is under an NDA that I haven't signed.

Promise you won't laugh at the ActiveX security provisions described at http://www.microsoft.com/office/ork/2003/seven/ch23/SecA05.htm. The Windows XP Security Center can also be spoofed quite easily, as mentioned at http://www.pcmag.com/article2/0,1759,1639276,00.asp.

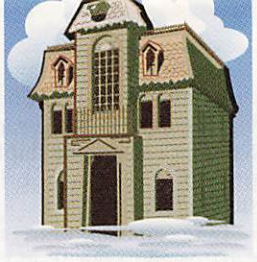
I used *convert -size 128x128 gray:infile outfile.png* to produce Figure 1. Linux systems spout random numbers from /dev/random or, if you don't mind slight correlations, /dev/urandom.

Cory Doctorow's presentation on why Digital Rights Management can't possibly work, given to Microsoft Research earlier this year, is at http://craphound.com/msftdrm.txt.

An RFID blog with useful links is at http://www.rfid-weblog.com/.

An encrypted disk program and a good overview of encrypted filesystems in general is at http://phk.freebsd.dk/pubs/bsdcon-03.gbde.paper.pdf.

DDJ



Eye On the Prize

Jerry Pournelle

The big news recently was the ANSARI X PRIZE (http://www.xprize.org/) was won by SpaceShipOne. Now I have no doubt that all but a few *DDJ* readers find this interesting and exciting, it may be less so obvious why it's particularly important to the computer community.

First, of course, it was paid for by one of our own—Paul Allen (and built by Burt Rutan). When the computer revolution began, IBM's vision of the future was 10-to-100,000 big central computers, all operated by white-coated technicians, serviced by IBM Customer Engineers, and running software leased from IBM or an authorized IBM partner. Bill Gates and Paul Allen had a different vision: A computer on every desk, and in every home, and in every classroom. While it is likely that Moore's Law would see to it that the great computer revolution took place without Microsoft, it's pretty certain that it wouldn't have happened so quickly without that vision.

We forget that a decade after the introduction of the PC there were serious debates about how much education it took to become computer literate, and computer literacy instruction techniques were taught in our colleges of education. UNIX gurus battled over the purity of programming, there were language wars, and many thought that while computers in homes and offices were useful, they wouldn't become ubiquitous because they weren't user friendly. That battle has long been settled, but it wasn't obvious to all that things would go as they have.

Even before they became ubiquitous, small computers had their effect in bringing down totalitarian states. Arthur Koestler once predicted that a sufficient condition for the dismantling of totalitarianism would be the free exchange of ideas within a totalitarian state, and the advent of the small computer made it certain that computer users would be able to freely communicate. The Falkland Islands War made it obvious to everyone that economic and military power depended on using small computers, while western grand strategy put more and more pressure on the USSR to modernize its weapons at great economic costs. The choice was clear: Adopt computer technology and let your intelligentsia communicate freely, or forfeit all pretense of military and economic power.

The USSR accordingly collapsed, and China changed irrevocably into something other than what it had been (although staying uniquely Chinese). It was not, as Francis Fukuyama rather precipitously proclaimed, the end of history, but the totalitarian state built around total control of information flow did slide into the dustbin of history and is unlikely to emerge short of a new Dark Age. The computer revolution accomplished at least that much.

Some years ago Esther Dyson's keynote at a meeting of the American Association for the Advancement of Science proclaimed the end of "the official story." Information monopolies don't work. If the story is false or leaves out a lot, the corrections will appear, and quickly. We have recently seen many instances of that.

The World Wide Web has stitched the entire civilized world into one enormous information pool; and the Web itself is evolving techniques for sorting information from misinformation. That process isn't anything like perfect, but it is in development.

Simulation

SpaceShipOne was paid for by money Paul Allen earned from the computer revolution, but that was hardly its main debt to the little beasts. The ship was designed with CAD programs running on off-the-shelf PCs. More importantly, though, all the flight simulation was done by a bank of networked off-the-shelf desktop Personal Computers, mostly from Dell.

SpaceShipOne made only six powered flights. Each one went higher and faster than the previous flight, and the last three reached space, the first commercial ship ever to do so.

While the ship itself flew only six times, the pilots had made hundreds of flights using highly realistic flight simulator programs. It was the only way they could learn to fly the ship.

In my early days in Projects Mercury, Gemini, and Apollo, flight simulators were expensive. Indeed, airline flight simulators cost almost as much per hour to operate as actual airliners. Computers were expensive to buy and expensive to operate. Now nearly any reader can afford a SpaceShipOne flight simulator. The actual software contains some proprietary information that Vulcan Ventures (the closely held company through which Paul Allen financed this effort) hasn't released, but the actual software isn't complicated and was built around standard programs. It runs on standard Windows machines using commercial video cards, and costs no more to operate than any other PCs.

The flight simulator was important for another reason.

SpaceShipOne's flights into space weren't all smooth. The first flight to space on June 21, 2004, began with a severe yaw anomaly and experienced some damage to nonstructural components. That was simulated and analyzed in the flight simulators. The second flight to space—first flight of the X PRIZE competition—was marked by a roll anomaly, with the ship rolling 29 times on its way up. That too was analyzed and simulated. The third flight to space was flawless as it won the ANSARI X PRIZE.

It couldn't have happened without the simulators, and that kind of simulation wasn't affordable to small companies before the computer revolution.

False Start

Some years after Apollo successfully landed on the Moon I had an argument with the late John J. Pierce, then chief technologist for the Jet Propulsion Laboratories (http://books.nap.edu/html/biomems/jpierce.html). Pierce reminded us that Heinlein's original fictional path to space led through suborbital flights of reusable vehicles; this developed both institutions and infrastructures, ways of making money from suborbital and then orbital flight, communications and control networks, and so on. Only then could space travel develop.

At the time, I thought Pierce mad. After all, we had just gone to the Moon, and I was proud to have been part of Project

Jerry is a science-fiction writer and senior contributing editor to BYTE.com. *You can contact him at jerryp@jerrypournelle.com.*

Apollo. Now, 30 some years later, I see he was right. Apollo was a government funded stunt, and didn't lead us in the right direction. Perhaps it couldn't. Now, though, we have the computer revolution, and we are on the right path at last.

Gigabit Connections

If you don't have gigabit Ethernet, you soon will—and it won't cost much. It's another of the fallout benefits of Moore's Law.

When we rebuilt Chaos Manor back in 1985 we designed the entire upstairs around computers. Of course, we got most of it wrong. For example, when we designed the new office suite, the best printer for a writer was still a machine that looked like an enormous typewriter and printed out manuscripts by impact: They came in several flavors, but in every case they printed on fanfold paper and worked at about 1200 characters a minute.

The advantage over the more usual dot-matrix printer was that despite the ragged edges of the fanfold paper, the manuscript looked like a typewritten manuscript, and that was what publishers were used to and some of them demanded. Dot matrix looked like computer printing, and was generally harder to read.

It took all night to print out the manuscript of a novel. The noise was deafening. Robert Heinlein once had a glass cage built around all but the keyboard of his IBM Executive electric typewriter because "I didn't want to be in the same room with a machine gun." That wasn't an option with the Diablo, later NEC Spinwriter, that I relied on for printing, so we designed a small closed room to hold the printer. It was originally intended to have a glass window in the door so you could see print operations in progress.

By the time construction was actually under way, Hewlett-Packard had come out with its LaserJet printers. They were so scarce there was a half year waiting list for them, but one of Dr. Bill Godbout's friends, Steve Rose, managed to order six of them, and Dr. Godbout talked him into letting me buy one. It came during construction, and was installed downstairs near my desk, and worked so silently that the "printer room" wasn't needed. We quickly modified that room to have its own small air conditioner, and wire channels to all the other parts of the house. It became the infamous "cable room" where we stored spare cables as well as where we kept the central server. Of course in those days of very early PCs, very early Macintosh, and the more robust and generally preferred S-100 bus machines, "server" had a different meaning.

So did networking. In 1985 the Ethernet chipset cost about $800, and that price looked stable. Bill Godbout and others in those days recommended ARCNET, because although it was slower and less technically elegant than Ethernet, it was a lot cheaper. The result was that I still have ARCNET coax pulled through various wire channel guides to the cable room. Needless to say, all that has been "abandoned in place." (Interestingly, ARCNET is still around, although it isn't all that popular. http://www.arcnet.com/.)

A few years later Artisoft and others brought the cost of Ethernet down from a few hundred to a few tens of dollars per system, and for a while we had Ethernet Thin-net coax cables, mostly strung overhead in what Roberta calls "computer decorator unfriendly" patterns. That worked at 10 megabits/second, faster than the old ARCNET. Moore's Law continued its inexorable pace, and 100 megabits/second Ethernet became available. It worked only on Category 5 twisted pair cables, but unlike Thin-net it would network in many different configurations. Those were the days of Ethernet "concentrators" (also called "hubs") that had both Thin-Net coax and twisted pair RJ-45 connectors. Over time those became switches, able to connect to both 10 and 100 megabit/second Ethernet connector cards. About then I had some of Alex's professional associates come in and pull Category 5 twisted pair Ethernet cable through much of Chaos Manor. That has served ever since.

Ethernet at 1 gigabit/second has been around a while. My last few Intel motherboards have gigabit Ethernet, but I paid little attention until one day I noticed a sale at Fry's: AirLink 5-port 10/100/1000/megabit Ethernet switches for about $25.00 on sale, with an 8-port for about $50.00. I had no idea how well those would work with my Category 5 cable system, but it would be cheap enough to find out, so I bought two 5-port and one 8-port switch.

When I got them home I opened the 8-port switch first, and discovered that it had to sit on a shelf: There were no little mounting holes on the back side of the case, as you will find on switches from Cisco, Hawking, and NetGear. Since the place I needed the gigabit switch required that I hang it on a wall, this was dismaying, and I left off doing anything with it until I could figure out where to put it. Then I got ready to go on a trip: One phase of this requires transferring a 1.6 gigabyte PST file to the laptop and sending another copy of that file to a backup storage machine. That takes about 11 minutes, and while my TabletPC doesn't have gigabit, the other systems do, and cutting the time to do either of those operations would be worthwhile.

When I opened the 5-port AirLink Ethernet Switch, I found it has mounting holes on the back. Only the 8-port lacks them. I quickly hung the 5-port on the bookcase where my other central concentrators and switches reside, and connected all the machines I thought capable of gigabit operations to it.

The AirLink switches have lights for each port to indicate what speed the device sees at the other end of its cable. I wasn't surprised to see that Lisabetta (the TabletPC) didn't have gigabit, but I was disappointed to see that Sable, an Intel D850EMVR2 dual-channel 3-GHz Northwood, was only 100 megabits/second. Anastasia, an Intel D875PBZ Northwood at 3.2 GHz, and Lassie, an Intel D865GBF Northwood, both had the gigabit lights, and later tests showed that all the Prescott systems have gigabit. Apparently my D850 was among the last to employ 100-megabit Ethernet chipsets.

When you go from 100 to 1000 megabits/second, you don't get 10 times better performance, of course. It's more like faster by a factor of three. Actual performance varies all over the place. With two machines connected through the same switch with no other tasks running in the background, the same 1.6 gigabyte Outlook.PST file transferred once in 263 seconds, and another time in 338, at 100 megabits/second. The same file took 135 seconds (first trial) and 71 seconds (second trial) transferring to a 1-gigabit machine. In neither case did CPU usage go above 25 percent, and the disks were large and nearly empty on both machines.

I have no explanation for the variance in times. Bob Thompson speculates that the Windows filesystem and TCP/IP stack have a lot to do with it. Anyway, it's clear that if you do a lot of big file transfers it's worth changing to gigabit switches if you already have gigabit capabilities in your machines.

Upgrading with Ethernet Cards

Note that it is probably not worth the effort to replace your current Network Interface Card (NIC) with a gigabit card. Gigabit Ethernet is enough to saturate the PCI bus, so simply adding a Gigabit Ethernet PCI Card is less than optimum practice, particularly for servers (where you would really want Gigabit Ethernet).

Intel motherboards take the Gigabit Ethernet adapter off the PCI bus and put it on a CSA (http://www.buildorbuy.org/csa.html) or PCI-Express bus. The right way to upgrade to Gigabit Ethernet is to replace the motherboard. That may be a bit more than most of us want, although I probably will do that for my older servers.

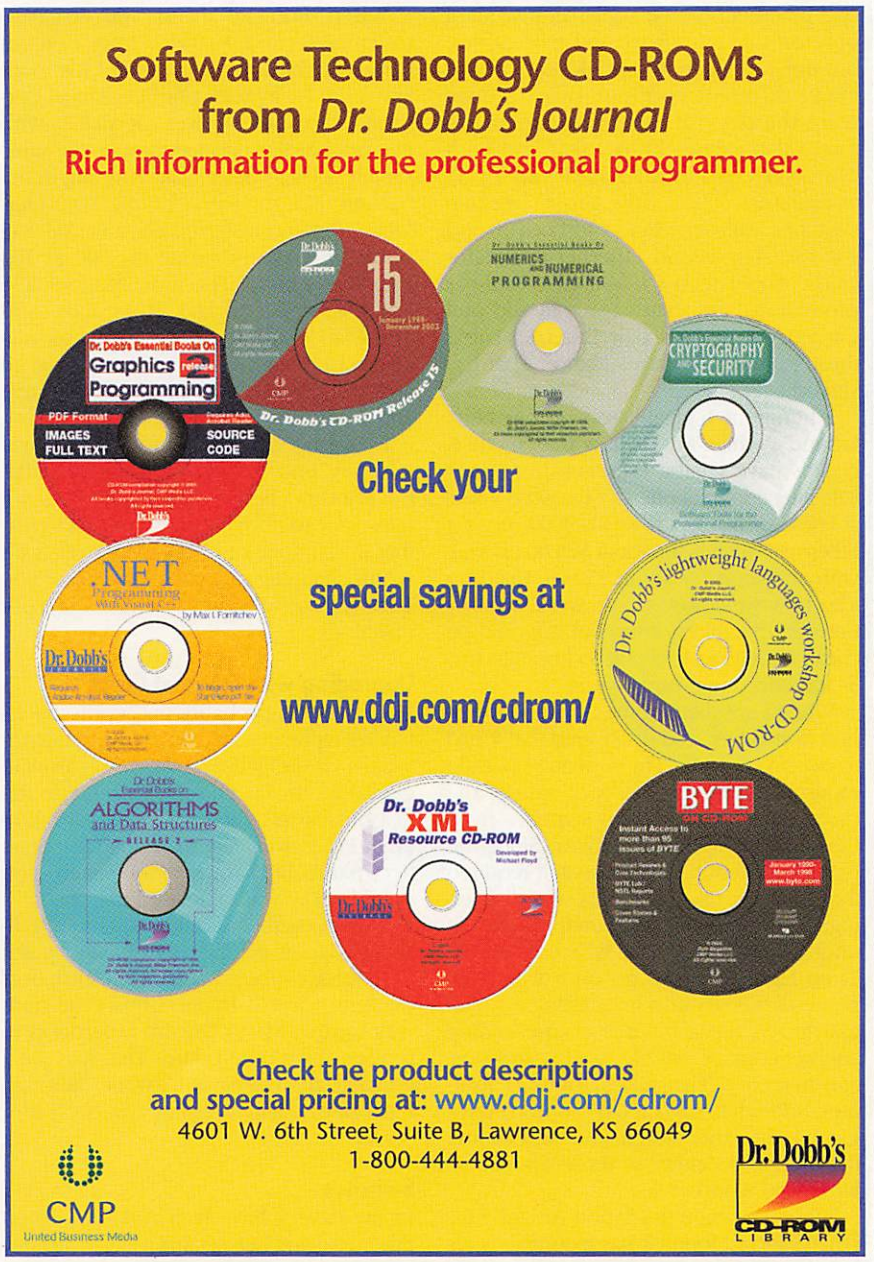
Switches

In my case, I have four AirLink Gigabit Ethernet switches set up: two at various

nexuses (nexii?) where my fastest machines connect to each other; another in the cable room; and a fourth out at the test stand where we set up new machines. The one in the cable room connects the others using the Category 5 cable we pulled several years ago.

Ethernet switches work this way: If two machines connected through the same switch want to talk to each other, the switch connects them without routing the signal anywhere else. If they are not both connected to the same switch, the switch sends the signals up the line to a switch that can find the system they're intended for. I have tested my gigabit connections both direct (two machines connected through the same switch) and indirectly (system to switch, switch to another switch then to a third switch, final switch to second system) and if there's any systematic difference in file transfer speed it's lost in the noise of other factors causing speed variance. Understand, I haven't been using any sensitive tools, just sending big files and recording the time it takes to transfer them.

My conclusion is that you'll get from 2.5 to 5 times the speed using Gigabit Ethernet (1000 Base T) rather than 100 megabit Ethernet (100 Base T), at least if you simply change to faster switches and you began with good quality Category 5 cable to begin with. Given that it's pretty cheap to do this, it was worth it for me. Note that Category 5 cable isn't rated for Gigabit Ethernet use, but as a practical matter it seems to be working; of course I have no wires longer than about 20 meters.



Category 5 and Category 6 Ethernet Cable

While I was at Fry's looking for something else, I decided to get some short Ethernet cables and thereby cut back on the rats' nests. I discovered that they sell 3-foot Category 5 cables for $4.99, and the same length Category 6 for $3.99. This seemed confusing because I would have thought Category 6 was "better" than Category 5; but no one at Fry's understood why they were priced that way, and for three cables I didn't really care about the price difference, so I bought Category 5 which I knew would work.

I should have bought Category 6, as you will find if you go to http://www.lanshack.com/cat5e-tutorial.asp where more is explained about Ethernet cables than most of us really want to know. Dan Spisak thinks there's a big push to get people using Category 6 cables and that's why the price break, but that's a guess. I doubt it made any difference, but in future, I'll get Category 6 cables, and if they speed things up I'll let you know.

Winding Down

The game of the month is a lament. I sure miss a lot of the old turn-based strategy games, particularly the battle simulations from SSI. The good news is that DOSBOX (http://dosbox.sourceforge.net/news.php?show_news=1) gets a number of those running properly. The bad news is that I gave away a lot of those SSI games that I wish I had kept, and I'll probably have to go haunt eBay to buy new copies now that I have ways to run them again.

The book of the month is *Who Are We?: The Challenges to America's National Identity,* by Samuel P. Huntington (Simon & Schuster, 2004). It's a deep inquiry into America's national identity crisis and ought to be required reading for anyone who is serious about self government.

The computer book of the month is David Pogue's *MAC OS X: The Missing Manual* (O'Reilly & Associates, 2003). If you want to connect your Mac to a Windows active directory system, you'll need this. Actually, you'll need this book for almost anything you want to do with a Mac; it really is the book that should have been in the box. If you have no interest in Macs, get Ken Milburn's *Digital Photography Expert Techniques* (O'Reilly & Associates, 2004), which is massively illustrated, readable, and full of information about taking better pictures with digital cameras. I only wish I had the time and patience to master this book; but just thumbing through it now and then helps.

DDJ

GPS.NET

Develop precise, efficient, and reliable location-based services with this multi-threaded framework for .NET & ActiveX!

- MAKE YOUR OWN GPS APPLICATION IN MINUTES
- AUTOMATICALLY DETECTS GPS DEVICES
- WORKS WITH ALL .NET FRAMEWORKS, EVEN 2.0!
- OVER 20+ CALCULATIONS FOR LAT/LONG & UTM!
- REAL-TIME GPS SATELLITE VIEWER

Try it FREE for 30 days!

www.gpsdeveloper.com

HEAVY METAL SECURITY

(Actual Size)

The USB device that offers ironclad security for your digital business

MARX Software Security

1-800-MARXINT info@marx.com www.marx.com

Order your free trial today at www.marx.com/dd

Build a database free of bad addresses

CASS certified address validation

"This solution was easy to implement. In fact Melissa Data supplies examples of the code on its web site. I was able to implement the production solution in a single weekend." *Mike Wheeler Chief software architect, Total Card Inc.*

Download a free trial at Melissadata.com/dobbs

1-800-MELISSA MELISSA DATA

DocJet

Source Code Documentation System

Need top-quality API documentation?

When you ship an API, you know that good documentation can be the difference between happy users and a very busy support line. DocJet is the only tool that builds complete documentation you can proudly ship to your customers.

If your documentation looks like something you just threw together, people aren't going to take it seriously. DocJet gives you **complete control over your output,** creating a docu-ment that matches your company's style guidelines.

DocJet supports single-source production of **printed output as well as all major on-line formats** - not just HTML. DocJet's MSHelp2 output can be integrated into Visual Studio .Net's help system.

DocJet uses the same ASCII formatting that we have been using to write comments since we got into the business - not some scheme cooked up in a cube. DocJet's simple approach to commenting makes it easy and rewarding to write better, more complete comments.

FREE TRIAL VERSION! www.talltree.com

TALL TREE Software Company

Results MATTER.

- High performance components for ISV and enterprise applications
- Analytics for finance, life sciences, etc.
- Windows, Unix, Linux, MacOS
- Java, C/C++, .NET, Fortran
- 64-bit versions

NAG Results Matter, Trust NAG

Numerical Algorithms Group

630.971.2337

www.nag.com • info@nag.com

tough environments demand eXtreme moves

Move to eXtremeDB™ the in-memory database with

- Transactions measured in micro-seconds
- Footprint approximately 100K
- High Availability version

New Transaction Logging version

Photo © A. Zinger

mcobject® precision data management

www.mcobject.com

Phone: 425-831-5964

D. E. Shaw Research and Development

Research on Algorithms and Architectures for Computational Biochemistry

Extraordinarily gifted computer scientists, systems architects, electrical engineers and systems software professionals are sought to join a rapidly growing New York–based research group pursuing an ambitious, long-term project aimed at achieving major scientific advances in the field of biochemistry and fundamentally transforming the process of drug discovery. This research effort is being financed by the D. E. Shaw group, an investment and technology development firm with approximately US $8 billion in aggregate capital, and operates under the direct scientific leadership of its founder, Dr. David E. Shaw.

Among the group's current research activities is a project aimed at developing a massively parallel special-purpose supercomputer and innovative mathematical and computational techniques to direct unprecedented computational power toward the solution of key scientific and technical problems in the fields of molecular simulation and molecular design. Successful candidates will be working closely with a number of the world's leading computational chemists and biologists, and will have the opportunity not only to participate in an exciting entrepreneurial venture with considerable economic potential, but to make fundamental contributions within the fields of biology, chemistry and medicine.

Serious candidates will have an exceptionally distinguished history of academic and/or industrial accomplishment in computer science, electrical engineering, applied mathematics, or a related area. Particularly relevant areas of expertise might include parallel computation, high-speed interconnection networks, scientific computing, numerical analysis, optimization, the analysis of algorithms, operating systems, digital systems simulation, reconfigurable computing, and ASIC design, but specific knowledge of any of these areas is less critical than exceptional intellectual ability and a demonstrated track record of achievement. We are prepared to reward exceptionally well-qualified individuals with above-market compensation.

Please send your curriculum vitae (including list of publications, thesis topic, and advisor, if applicable) to career67@desrad.deshaw.com.

D. E. Shaw Research and Development, L.L.C. does not discriminate in employment matters on the basis of race, color, religion, gender, national origin, age, military service eligibility, veteran status, sexual orientation, marital status, disability, or any other protected class.

D E Shaw & Co

Fast Flexible Powerful

Build a Parser in 15 minutes - YES, 15 MINUTES!

Slash time and have fun developing
- scripting languages
- compilers
- assemblers
- data parsers

Free Trials!

Introducing Dr Parse™
Smart Parsing Technology™

- Multi-Language - DLL
- VIP - Visual Interactive Parser (Dr Parse VIP only)
- PFC - Parser Foundation Classes (30,000+ lines of C/C++source)
- PFC is completely Object Oriented
- Printed Manuals
- Competitive Upgrade & Academic Pricing
- 30-Day Money Back

Get Your Copy Today
www.zpnetics.com
Toll Free: 888-ZpNetics(888-976-3842)
Additional Products
Secure Online Ordering
© 2004 ZpNetics, Inc. All Rights Reserved
ZpNetics

software development 14th annual productivity award

Build Easy.
Build Fast.
Build Smart.

Visual Build PROFESSIONAL

Want to streamline your build process? Visual Build Professional makes it easy for software professionals to create an automated, repeatable process for building and deploying quality software. It's the powerful, affordable build solution that integrates fully with Microsoft Visual Studio and works with virtually any development tool. To learn more or to try a free evaluation copy, visit our website.

www.visualbuild.com

kinook

CodeMax

Powerful Code Editing in a Windows Control

- Color Syntax Highlighting
- Custom Language Support
- Drag-and-Drop Manipulation
- Over 150 Unique Edit Commands
- Custom Keystroke Assignments
- Multiple Splitter Views
- Unlimited Undo/Redo
- Persistent Keystroke Macros
- Auto-Indentation
- Bookmarks
- Column Selection
- ActiveX or DLL Versions, .NET Compatible
- MFC Support
- Win95/98/ME, NT4, Win2K, XP, .NET Server
- Includes 300+ Page Reference Manual

$199!

Source Code Available!

WinMain Software
http://winmain.com
sales@winmain.com
Fax: 425-338-2638

DEVELOPERS' CHOICE AWARD
EDC Evans Data Corporation
2004
Most Trusted Publication for Tools and Software Technologies
Dr. Dobbs Journal

We at *Dr. Dobb's Journal* would like to thank the developer community for choosing us as the most trusted developer publication .... again!

CMP
Dr. Dobb's JOURNAL SOFTWARE TOOLS FOR THE PROFESSIONAL PROGRAMMER

Software Technology CD-ROMs from *Dr. Dobb's Journal*

Rich information for the professional programmer.

Check the product descriptions and special pricing at: www.ddj.com/cdrom/
4601 W. 6th Street, Suite B, Lawrence, KS 66049 • 1-785-838-7500

Dr. Dobb's CD-ROM LIBRARY

Exceptional C++ Style

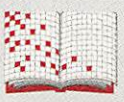
Matthew Wilson

Exceptional C++ Style is the third book from Herb Sutter, and the third in the *Exceptional C++* series, following the eponymous *Exceptional C++* and its sequel *More Exceptional C++*.

I should say upfront that *Exceptional C++* is my favorite C++ book. When the book came out, it was like someone switched on a light on post-C++-98 C++ practice. It had the clarity of writing of Scott Meyers, (almost) the efficiency of Bjarne Stroustrup, and accessibility to difficult issues previously unseen. Although I've read a lot of highly recommendable books since then, none has matched that impact.

Exceptional C++ Style: 40 New Engineering Puzzles, Programming Problems, and Solutions starts off well with a look at (ab)uses of *std::vector*. As usual, Sutter manages to tease out abuses that you'd probably never dream of, but that are instructive nonetheless. If you're like me, your craft is moderated and governed by experience, and lessons learned quickly moved out of your consciousness into your subconscious instincts and style. I think one of the values of the *Exceptional C++* books is that they bring such lessons to the fore. Either Sutter gets to see a lot of bad code, or he's able to put himself into mindsets that practicing programmers work hard to leave behind. Whichever, it's a great foundation for his educational writings.

Items 2 and 3 concern themselves with *sprintf()*, *snprintf()*, string streams, and Boost's *lexical_cast* as measures for converting from different types to strings. The material is largely up to scruff, but I had a problem with the broad-brush statements that *sprintf()* is unsafe/insecure, rather than identifying the actual causes of its susceptibility to overruns, such as being asked to translate types that are themselves of variable length. After all, there's absolutely nothing unsafe about converting any integral type to a string of 21 or more characters by *sprintf()*. I also thought it a bit too kind of Sutter not to object more strenuously to the poor levels of performance he'd ascertained for some of the more complex solutions. That's the kind of thing that gives C++ a bad name for being overly complex and inefficient.



Exceptional C++ Style: 40 New Engineering Puzzles, Programming Problems, and Solutions
Herb Sutter
Addison-Wesley Professional, 2004
352 pp., $39.99
ISBN 0201760428

Item 4 discussed member function adaptors, and gave me another justification for my antipathy to default parameters. Item 6 covers a common misapprehension with respect to pointers and iterators and the evil old *operator &*. It also offers a nice guideline to the effect that if your type has a *swap()* method, then you should overload *std::swap()* for it. This book is full of these golden nuggets.

Item 7 discusses the ramifications of function templates, specializations, and overloading. This is essential template survival guide stuff. Item 8's coverage of techniques for befriending templates leaves you yearning for better compilers.

Items 9 and 10 are about *export*. Here I had some real conflicts. The explanation given of *export* is nothing short of wonderful. However, Sutter sets you up to form your own conclusion that *export* is the most colossal white elephant in programming history, without actually making the point himself. I think he is wrong to leave you dangling so. Whatever the imagined virtues at the time of standardization (1995–98), to have something so unbelievably complex (as Sutter says, it took the finest template/compiler minds longer to make *export* work than to implement Java) in the language standard, which only one compiler currently provides, is nothing but perfect ammunition for C++'s detractors.

Being about exceptions, Items 11–13 are classic *Exceptional C++* and truly exceptional. C++ writing doesn't get any better than this! Item 14 covers virtual inheritance and construction order and, although it curiously fails to discuss diamond inheritance, was a good refresher; I found it odd to be forced to think about virtual inheritance, having not used it in years.

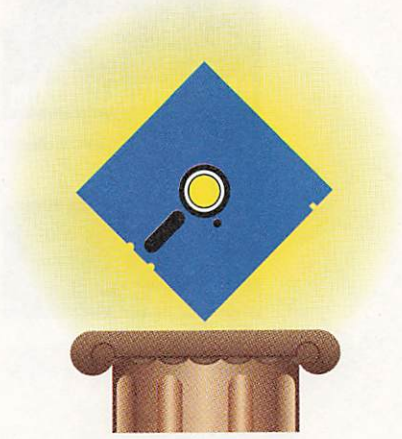
Item 16 is brilliant. While it didn't actually teach me anything per se, it enhanced and clarified both my experiential understanding, and cleared my terminology. Above all, this is what I value most in Sutter's writings. The same goes for Item 17. Item 18, which describes the NonVirtual Interface pattern, is also good, although it missed one advantage of NVI—how it clears up the default argument confusion you tend to get into with virtual methods.

The remainder of the book covers a variety of subjects, including memory, optimization, odd little puzzles, and style issues. It also takes a few swipes at *std::basic_string*, which is fair play in my book! There are a few dodgy ones, but the bulk of them are enlightening, entertaining, and educational.

Is *Exceptional C++ Style* as good as *Exceptional C++*? Maybe not. But it is a very good book, and you'll do yourself a disservice if you don't rush out and get a copy straight away.

DDJ

Matthew is the creator of the STLSoft libraries, and author of Imperfect C++ *(Addison-Wesley, 2004). Matthew can be contacted at matthew@synesis.com.au.*



IDELIX Software has released new versions of its Pliable Display Technology (PDT) SDK. PDT is a virtual lens technology for visualizing and interacting with digital data. PDT-based functionality such as detail-in-context data interaction, lens-assisted operations, bandwidth management, and multilayer viewing are designed for applications, such as geospatial intelligence, defense, GIS, medical imaging, gaming, and CAD. Version 3.1 of the PDT SDK is written in C++ and features new COM and .NET wrappers.

IDELIX Software
555–1122 Mainland Street
Vancouver, BC V6B 5L1 Canada
604-656-6300
http://www.idelix.com/

TechExcel has unveiled its DevTest test management solution, which automates test coverage creation, test planning, and scheduling, and tracking of test execution. Using DevTest, QA managers and engineers can create, manage, and document test plans in a centralized repository. Users can then schedule and assign these tests to their team using a wizard driven interface. Once assigned, testers execute these tasks, and defects can be automatically submitted to development projects based on test case status, establishing links between the test task and the submitted defects. DevTest tracks tests in built-in presentation quality reports and in a planning dashboard.

TechExcel
3675 Mt. Diablo Boulevard, Suite 200
Lafayette, CA 94549
925-871-3900
http://www.techexcel.com/

Toolsfactory has updated its Doc-O-Matic documentation system to Version 4. Doc-O-Matic uses information about source-code symbols (namespaces, packages, classes, members structs, records, enums, and other symbols), and existing tagged and untagged comments (XMLDoc comments, JavaDoc comments and plaintext comments) in source-code files and external documentation files (Word Documents, HTML files, and plaintext files) to create cross-linked documentation systems, online help and user manuals in PDF, HTML, HTML Help 1.x, Help 2.0, Windows Help, RTF, and XML. Doc-O-Matic 4 supports C/C++, IDL, C++.NET, C#, VB.NET, Delphi, JavaScript, ASP.NET, MATLAB, JSP, and Java.

Toolsfactory Software
1005 Terminal Way
Reno, NV 89502
866-515-8462
http://www.toolsfactory.com/

OpenFabric is a platform for building Internet-based distributed systems, including content-management and ASP systems. It comprises a distributed object repository and class dictionary, object replication, scripting engine, workflow engine, web-page designer, virtual filesystem, desktop client, and development environment. OpenFabric runs on Windows and Linux.

Suntail AS
P.O. Box 651
St. Albans, Herts AL1 4YZ UK
http://www.openfabric.net/

Parasoft has announced SOAPtest 3.0, an automated web-services testing tool. It supports WSDL schema verification and Standards-compliance checks; automatic test creation using WSDL and HTTP traffic; data-driven testing through data sources (Excel, CSV, Database Queries, and so on); scenario-based testing through XML Data Bank; scripting with Java, JavaScript, and Python; report generation in HTML, XML, and Text formats; and real-time graphs and charts. SOAPtest 3.0 features an improved UI and is available for Windows, Linux, and Solaris.

Parasoft
101 E. Huntington Drive, Second Floor
Monrovia, CA 91016
626-256-3680
http://www.parasoft.com/

ILOG is now packaging its JViews 6.0 graphics toolkit as standalone products. The ILOG JViews product line includes JViews Diagrammer 6.0, JViews Charts 6.0, JViews Gantt 6.0, JViews Maps 6.0, and JTGO 4.0. The updated JViews products now support the Business Process Management Notation (BPMN), a Standard that helps developers model business processes, as well as Java Standards like OSS/J (JTGO) for telecom operational support services (OSS) applications, and Java Server Faces technology for creating thin clients.

ILOG
1080 Linda Vista Avenue
Mountain View, CA 94043
650-567-8000
http://www.ilog.com/

Viklele Associates is offering the Runtime Form Designer Component Suite 1.0, for developing a form designer application similar to Visual Basic IDE. VLFormDesigner allows runtime resizing and repositioning of ActiveX controls placed on it; VLPropertyList provides COM introspection capability for creating a Visual Basic properties-view-styled user interface; and VLButtonBar is provided for creating a professional tool palette. Viklele Associates also offers VLMenuPlus, VLViewPort, and VLFullScreen.

Viklele Associates
8, Shree Vidya
Rajpath Society, Paud Road
Pune 411 038 India
+91-20-25381012
http://www.viklele.com/

Software developer and computer artist Vincent Stahl has developed the d3caster Game Engine, a 3D Java gaming development kit embedded within an open-source demo game. The package, intended for experienced Java developers, is designed to promote sound programming, fast image loading, image conversion, and 3D raycasting graphics with dynamic light effects — all made with pure Java, without the need for special plug-ins or hardware.

Stahlforce
An der Betteleiche 16
70569 Stuttgart Germany
http://www.stahlforce.com/

JBoss has introduced JBoss jBPM, a workflow engine formerly known as the "Java Business Process Management" (jBpm) project. Like all federated JBoss Professional Open-Source middleware, JBoss jBPM is a standalone product. It can be deployed on its own or with any J2EE application server. The engine uses the native jBPM Process Definition Language (JPDL), designed to support any Standard or specification, including BPEL, BPELJ, BPML, BPSS, ebXML, WSCI, and XPDL.

JBoss
3340 Peachtree Road, NE, Suite 1225
Atlanta, GA 30326
404-467-8555
http://www.jboss.com/

DDJ

Dr. Dobb's Software Tools Newsletter
What's the fastest way of keeping up with new developer products and version updates? Dr. Dobb's Software Tools e-mail newsletter, delivered once a month to your mailbox. This unique newsletter keeps you up-to-date on the latest in SDKs, libraries, components, compilers, and the like. To sign up now for this free service, go to http://www.ddj.com/maillists/.

The Advertiser Index

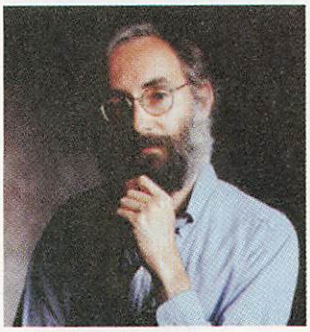
Visit Our Advertisers' Web Sites Today!

ADVERTISERS' NAME	PAGE #
Abraxas Software www.abxsoft.com	18
Altova www.altova.com	4
Amyuni Technologies www.amyuni.com	9
C/C++ Users Journal www.cuj.com/sub/	49
ceTe software www.dynamicpdf.com	44
CrypKey www.crypkey.com	18
D. E. Shaw & Co. www.deshaw.com	79
DevConnections www.devconnections.com	S7
dtSearch www.dtsearch.com	63
Enerjy www.enerjy.com	13
FairCom www.faircom.com/ep/ddj/ado	6
Game Developers Conference gdconf.com	43
Google www.google.com/ddj	20
GPS.NET www.gpsdeveloper.com	79
Guibot www.guibot.com	33
IEEE www.ieee.org/member2	67
InstallShield www.installshield.com/10.5_strong	7
Iocomp www.iocomp.com	S10
Kinook www.visualbuild.com	80
LEAD Technologies www.leadtools.com	19
LEAD Technologies www.leadtools.com	S5
LinuxWorld www.linuxworldexpo.com	31
Marx Software Security www.marx.com/dd	79
McObject www.mcobject.com	79
Melissa DATA melissadata.com/dobbs	79
Microsoft msdn.microsoft.com/visual	C2-1
Mindreef www.mindreef.com/free	51
Numerical Algorithms Group www.nag.com	79
O'Reilly www.oreilly.com/go/dobbs	2-3
OSDL Conference www.osdllinuxsummit.org	S15
Parasoft www.parasoft.com/achievequality	11
Pegasus Imaging www.pegasusimaging.com	41
Programmer's Paradise programmersparadise.com/ca	23
Programmer's Paradise programmersparadise.com/intel	25
Programmer's Paradise programmersparadise.com	27
Recursion Software www.recursionsw.com/cplus.htm	20
SafeNet www.safenet-inc.com/sentinel-500	C3
Scientific Toolworks www.scitools.com	55
SD West 2005 www.sdexpo.com	61
Smart Bear Software smartbearsoftware.com	47
Software FX www.softwarefx.com	S2-S3
TAL Technologies www.taltech.com	S9
Tall Tree www.talltree.com	79
TechNet Magazine technetmagsubs.com/?k=ddjjan	S13
Trolltech www.trolltech.com	C4
WIBU Systems www.wibu.com	74
WinMain Software winmain.com	80
ZpNetics www.zpnetics.com	80

The index on this page is provided as a service to our readers. The publisher does not assume any liability for errors or omissions.

Contact your Account Manager to discuss an ad campaign that best meets your advertising goals!

Visit www.ddj.com/ddj/adstaff.htm for contact information.



Grading On the Curve

As I write this, the U.S. Presidential election still hasn't been decided, so it's not clear yet how badly the polling pros have flubbed it by missing all those trendy people who no longer have land-line phones. Of course, the pollsters missed 10 times as many harried citizens who refused to pick up their land-line phones, refused to answer any questions, or attached eardrum-damaging noisemakers to their phones to discourage the incessant canvassing and polling calls. It's hard to believe that you can get a representative sample of the electorate by talking only to the bored and lonely, but I'm not a professional pollster.

I spoke vaguely just then of "those" people who have forsaken tethered phones for the cell-phone-only lifestyle, but I was taken aback to discover the other day that I have become one of them. When I mentioned to my partner Nancy that no messages had been left on our home answering machine for several days, she explained, "that's because I canceled our service a week ago."

Granted, her office and mine (with land-line phones) are just across the driveway from our house and we spend most of our waking hours in our offices, even eating most of our meals in the same building (Nancy's business is a restaurant). But I was still a little disoriented to have been inducted without my knowledge into this trendy group. I'm usually only ahead of the curve when it leads down a blind alley, like my quixotic detour into Newton software development.

Also, I'm supposed to be in charge of all technology in our relationship, just as Nancy is in charge of selecting the wine to have with dinner.

The whole no-land-line experience made me pause to assess where I fell on other current tech trend lines. Behind the curve? Ahead of the curve? Or in the ditch, flies buzzing around my eyes, blood on my saddle?

Take computers. I just bought a new desktop machine, and like the rest of the machines on our office network, it's as untethered as a first-time voter. Not only is it networked via Wi-Fi, its mouse and keyboard, Bluetooth devices both, rest in ghostly disconnectedness on my desktop. No monitor cable, either: The pretty display that Nancy's employees mistake for the monitor is the whole magilla. You've deduced by now that I'm talking about an Apple iMac G5. I know, owning a second-tier Apple computer loses me some geek points, but there's no denying that it is a trendy machine.

But in this post-PC era, personal computers are a poor place to look for real trendiness. After all, in a blatant example of trendy tail wagging dragging dog, Apple markets this computer as being "from the makers of the iPod." And do I have an iPod? No. I hardly have any portable devices at all, except for a Wi-Fi finder, which lets me feel like a warblogger. But I'm not a warblogger, and I've downloaded exactly one song in the past year, which I paid for. Not trendy.

It's even harder to stay trendy in one's online presence. I turned my personal web site into some semblance of a blog a while back, but unless you're running a wiki, you're hopelessly behind the curve, so color me behind. I think ideally I should be podcasting, if I knew exactly what that was.

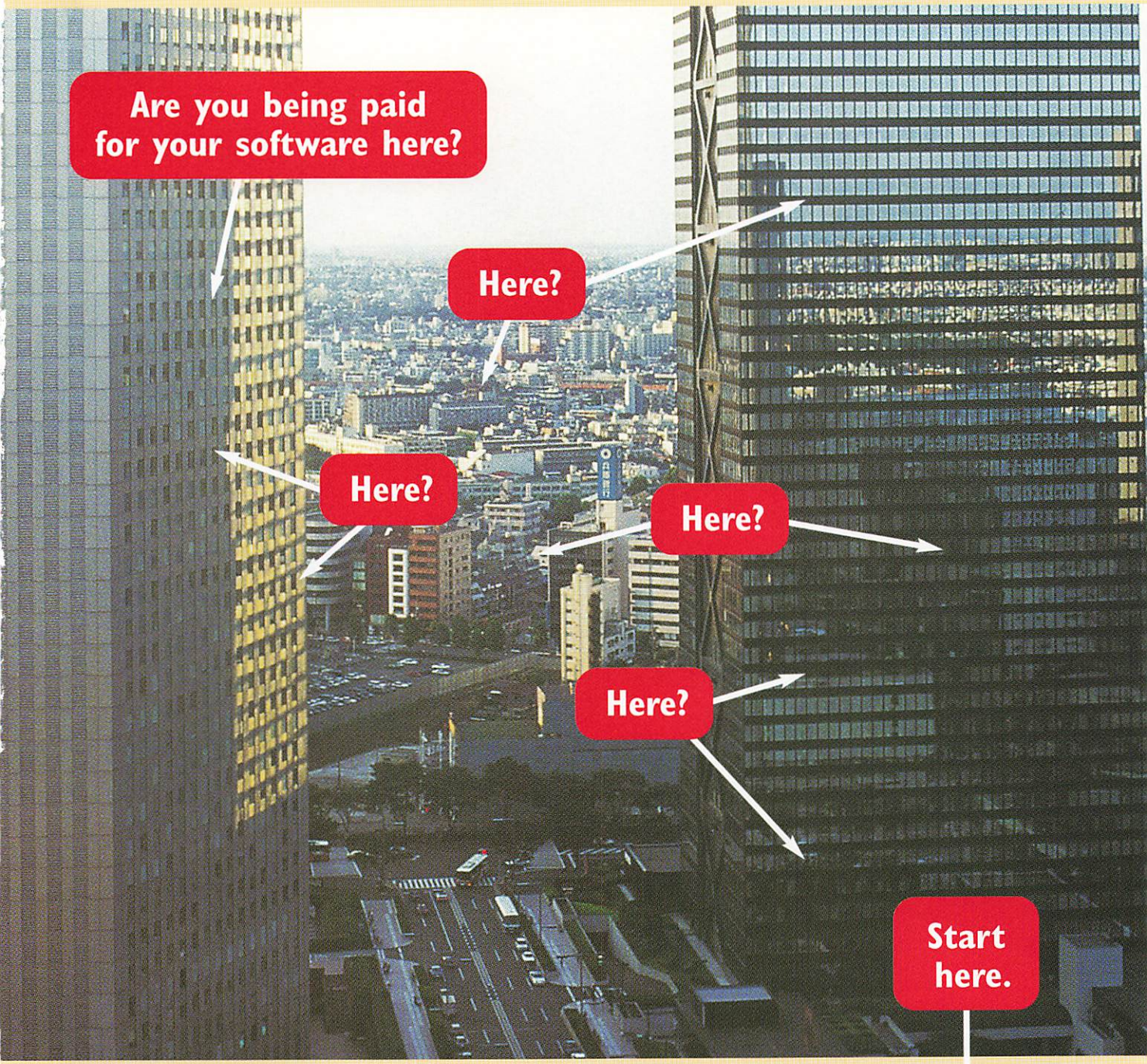
Then there's television. Television is trendy again—not watching it, but owning the hot hardware. But my household is stuck back in the 20th Century, TVwise. No DTV, no plasma screen, no TiVo. We don't even have (can I really admit this?) a DVD player. We can burn 'em, but we can't play 'em. On the other hand, ours is a satellite-only household: We haven't watched broadcast television in years.

Does any of this really matter? Over dinner I asked Nancy if she thought we were missing anything by not embracing all the latest trends. As she mulled it over, she unscrewed the cap from a New Zealand Sauvignon Blanc, and I realized that every profession has its trends to keep up with.

Michael Swaine

Michael Swaine
editor-at-large
mike@swaine.com

PS: Oregon is the only state in the U.S. in which everyone votes by mail. E-voting is the trendy techie way to vote, but the campaign trail is one road on which I'm happy to be behind the curve.



Sentinel

Protect your revenues as well as your software.

Are you giving away one copy of your software for every two that you sell? It's quite likely. Because a recent study shows that only two of every three software applications in use today were purchased legally.* So if you can't afford to give up a third of your revenue, get SafeNet Sentinel. We've been protecting software developers against piracy and hackers for more than 15 years. We'll help manage your rights – so you get what's rightfully yours.

Call 1-800-533-3958 to be SafeNet sure.
www.safenet-inc.com/sentinel-500

SafeNet
The Foundation of Information Security

Copyright 2005, SafeNet, Inc. All rights reserved. SafeNet and SafeNet logo are registered trademarks of SafeNet, Inc.
**First Annual BSA and IDC Global Software Piracy Study, July '04. (NASDAQ: SFNT)*

APPLICATIONS - AUTHENTICATION - REMOTE ACCESS - ANTI-PIRACY - LICENSE MANAGEMENT - VPN/SSL

THEY'VE GOT TWO THINGS
IN COMMON:
THE FREEDOM OF LINUX.
AND THE TECHNOLOGY THAT
BRINGS IT TO THE DESKTOP.
Linux is rapidly expanding to the desktop, bringing with it new opportunities – and demands for flexible, powerful and user friendly applications.
Qt® – the comprehensive C++ application development framework – provides technology that empowers software engineers to deliver native, commercial-grade Linux applications to the desktop. Applications that can also target the other major platforms – Windows, Mac, Unix – all from a single code base.
Enjoy the freedom of Linux on the desktop. Get Qt today.
Download your evaluation copy at: www.trolltech.com
Qt BY TROLLTECH
CODE LESS. CREATE MORE.
Trolltech, Qt, and the Qt logo are registered trademarks of Trolltech AS in Norway and/or other countries worldwide. All other trademarks are property of their respective owners.